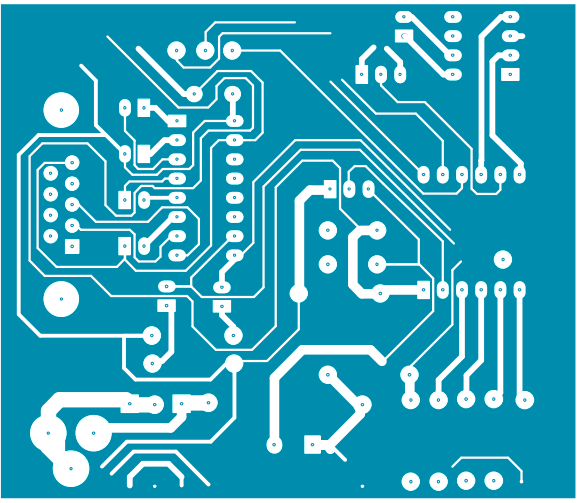
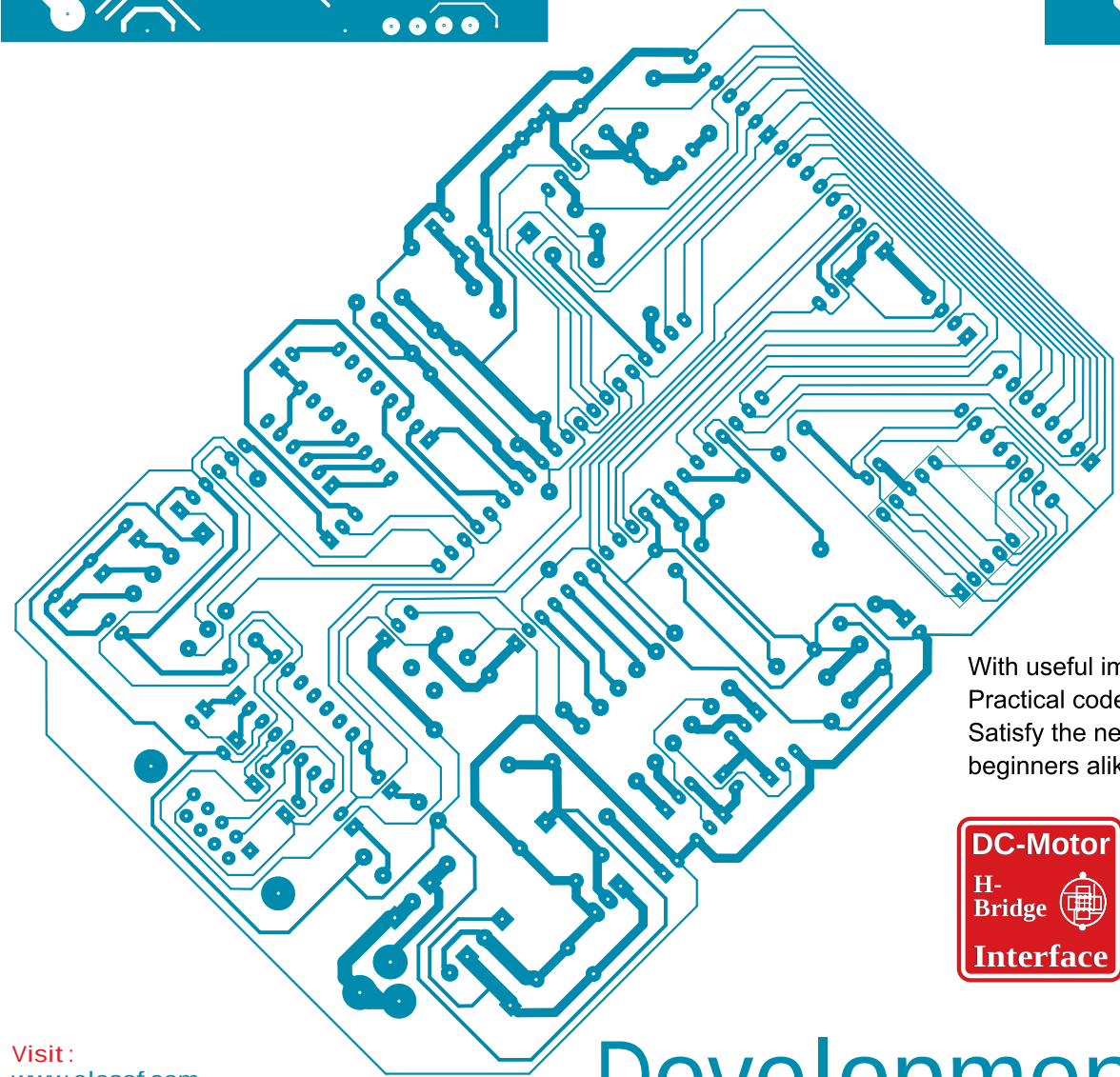
EleSof®

Enriching Life through Knowledge....

PICBox1®



USER MANUAL



With useful implemented peripherals, plentiful Practical code examples. EleSof development Satisfy the needs of experienced engineers & beginners alike.

DC-Motor H-Bridge Interface

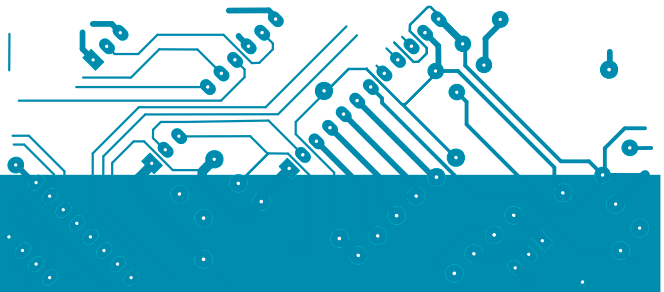
CONNECTOR 2x16 LCD ON-BOARD

7 SEG READY

Visit :
www.elesof.com
www.projectsmaker.com

Development System

EleSof®
Development System



TO OUR VALUED CUSTOMERS

I want to express my thanks to you for being interested in our products & having confidence in EleSof Technologies Pvt. Ltd.

It is our intention to provide you with the best quality products. Furthermore, we will continue to improve our product performance to better suit your needs.

EleSof Technologies Pvt. Ltd.

The Microchip® name and logo, PIC,® dsPIC® are registered trademark of Microchip Technology Incorporated in the U.S.A. & other counties. All other trademarks mentioned herein are property of their respective companies and are only used for the purpose of identification or explanation and to the owner's benefit, with no intent to fringe.

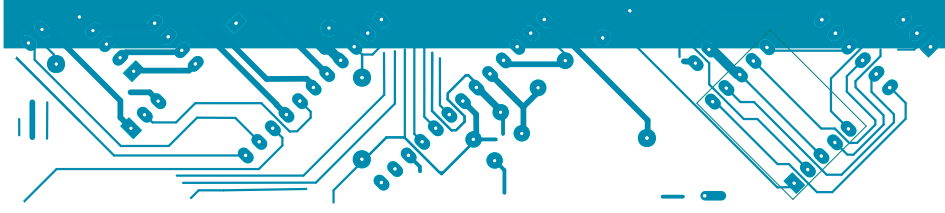


TABLE OF CONTENTS

Introduction to PICBox1 Development System……..4
Key Features……5
- ICSP Programming……..6
- Power Supply……..7
- LED's……...8
- Seven Segment……9
- Switch's……..10
- PWM……..11
- Buzzer……12
- DC Motor.....13
- Stepper Motor……14
- ADC……...15
- 16x2LCD Display…..……...16
- Rs232……17
- I^2C EEPROM……...18
- I/O Pins……19

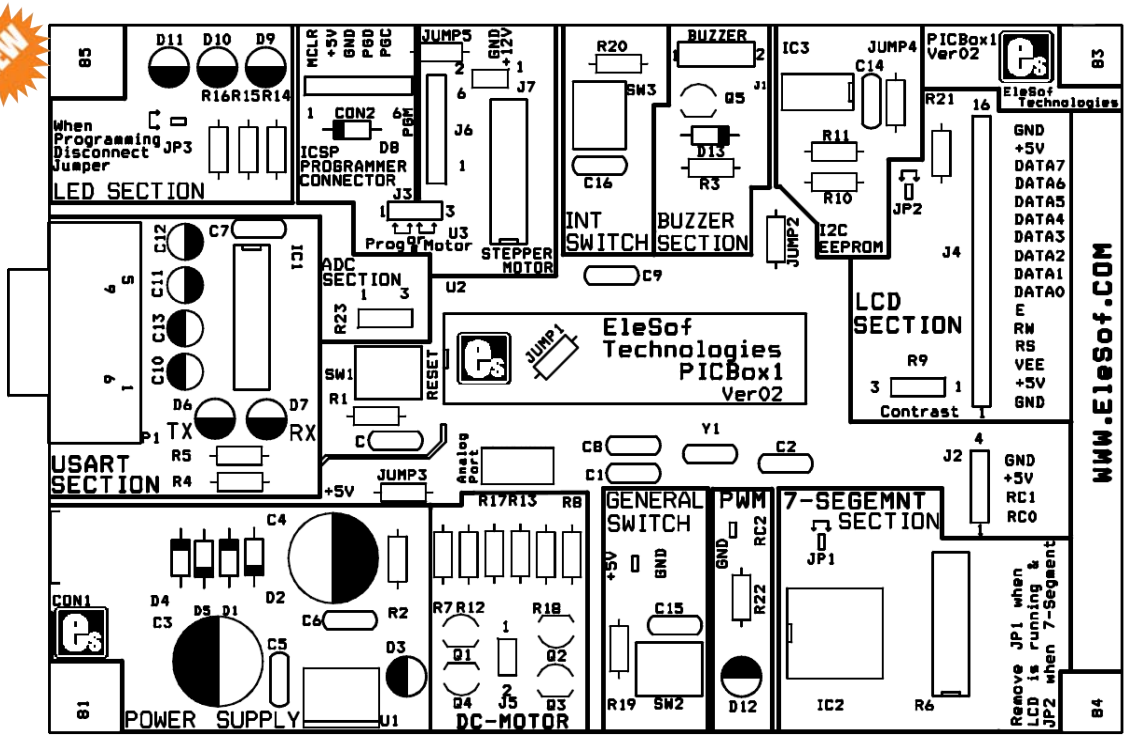
Schematic of PICBox1……..20
Introduction to Embedded Systems…….....21
Introduction to PIC and CCS Compiler…….....22-23
Overview of Embedded C……..……24-25

Experiments On Board
- Lab[1]: Configuring I/O pins……..……..…….26
- Lab[2]: Button Sensing……..…….....27
- Lab[3]: Analog to Digital conversion (ADC)…….....…..28
- Lab[4]: Seven Segment Based Timer…….....29
- Lab[5]: Drive 16x2-LCD Display at 8-Bit mode…..……...30-31
- Lab[6]: Rs232 Communication ……..……..…….....32
- Lab[7]: Read/Write Internal EEPROM……..…….....33
- Lab[8]: Read/Write External EEPROM ……..…….....34
- Lab[9]: PWM (Pulse Width Modulation)……...35-36
- Lab[10]: Drive Buzzer.....37
- Lab[11]: Drive Dc Motor……..…….....38-39
- Lab[12]: Drive Stepper Motor.....40-41

Introduction to PICBox1 Development System

The PICBox1 Pic16F8XX Ver01 Development Board is a versatile kit, which can be used as a prototype developer. This Kit is readily usable for your desired designs along with In-Built ISP programming Connector. The Development Board is the best way to get an indisputable acquaintance in PIC. In this board user can use any 40 pin Microcontroller of PIC16F8XX series. The RS232 driver on board allows easy connection with PC or other embedded hardware. The board has User button and status LED. The bridge rectifier allows this board to be powered with both AC and DC power supply adapters. You are simply expected to write a code in any compiler which support MICROCHIP controller, generate a.hex file and program your microcontroller using ICSP programmer without removing the controller from development board.

NEW



PIC DEVELOPMENT BOARD

CONNECTOR 2x16 LCD ON-BOARD

7 SEG 8. READY

ADC ON-BOARD

SERIAL EEPROM ON-BOARD

STEPPER Motor ULN2003 Interface

DC-Motor H-Bridge Interface

PICBox1 Specification & Features

MCU Devices

- PIC16F8XX.

Clock

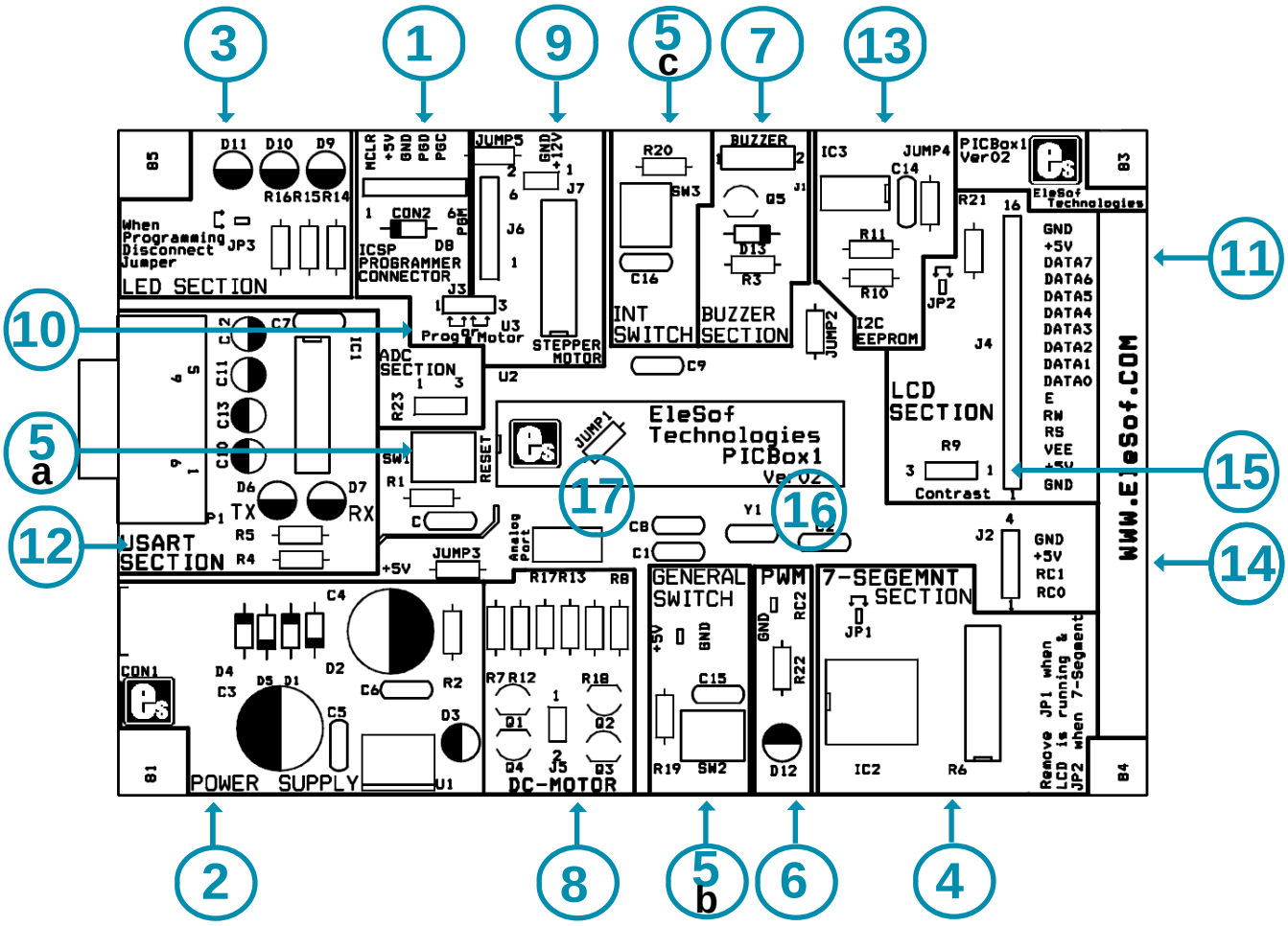
- 10MHz (0-20 MHz Maximum Clock Speed).

On-Board Features

- Reset Switch.
- Buzzer interface.
- Power plug-in jack.
- One I2C EEPROM.
- 7-Segment interface.
- Quartz crystal 20 MHz.
- 3V DC motor interface.
- 3-General purpose LED's.
- One LED for PWM section.
- DIL40 microcontroller socket.
- ADC Input Test (Potentiometer).
- General Input-Output Connector.
- Tactile Switch for Interrupt Study.
- +5V power supply voltage regulator.
- 8 Bit 16x2 Alphanumeric LCD interface.
- Tactile Switch for Switch Function Study.
- Single layer PCB Dimensions is 6x4 inch.
- ICSP connector for programming using serial port.
- Unipolar Stepper motor interface with ULN support.
- RS232 Tx, Rx interface with MAX232 IC on socket with Tx, Rx indicator LED.

PICBox1 Package Contain

- Buzzer.
- Dc Motor.
- 16x2 LCD.
- Stepper Motor.
- RS232 Serial Cable.
- 9V AC Power Adaptor.
- Single Side Routed Pic16F8XX Ver01 Development Board with 16F877 Microcontroller & discreet component.
- CD: Examples circuit & Codes, Datasheet's, ICSP Software, Demo CCS Compiler & User Manual's.

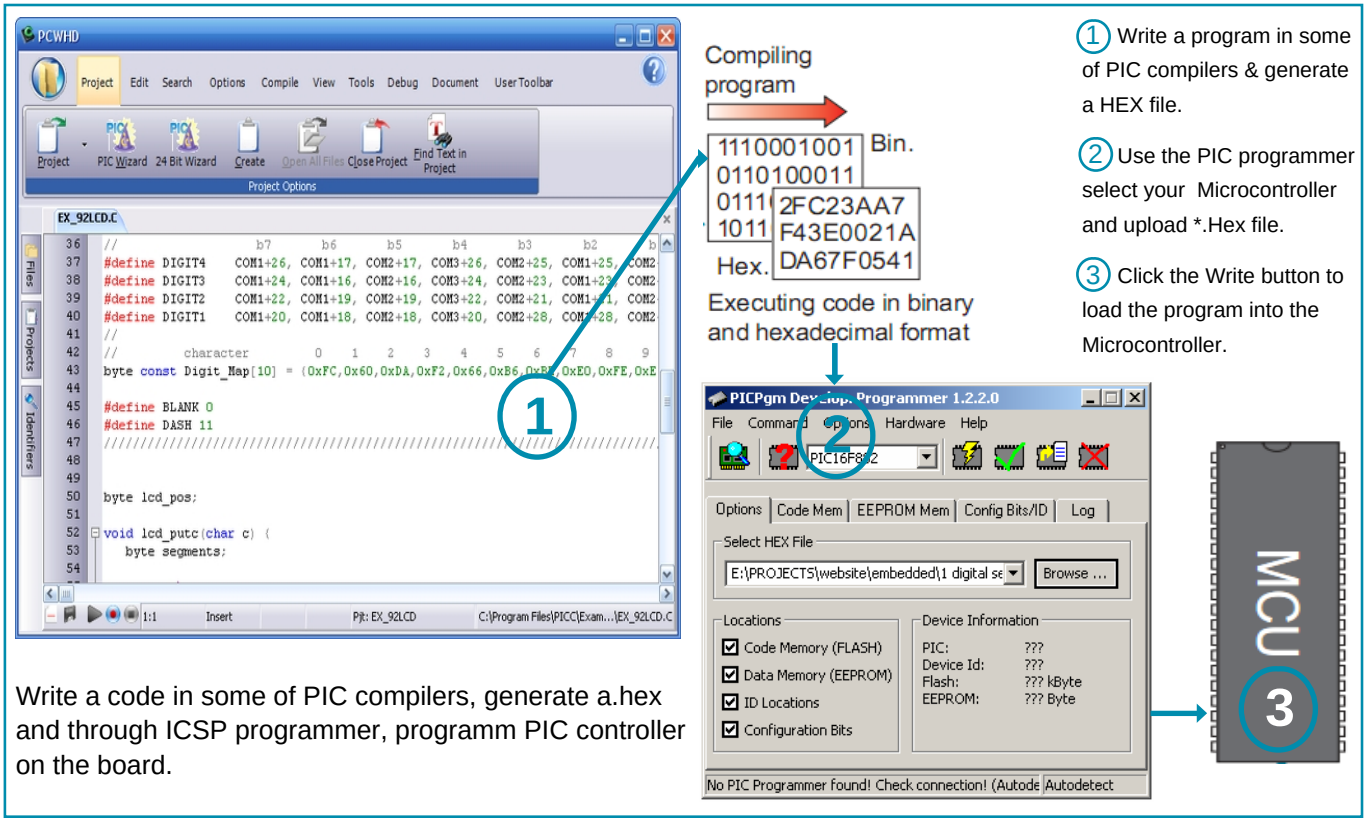


Key Features

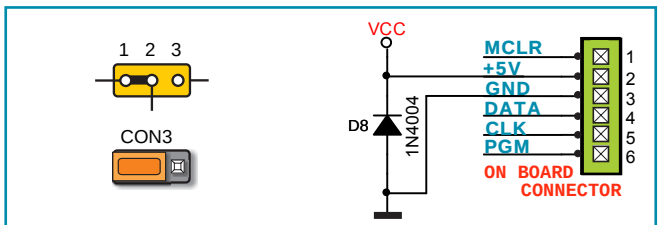
1: 6 Pin ICSP Programming Connector.
2: +5V Power Supply Support DC/AC Both Adapter.
3: LED's to Indicate Pin's Logic Stage.
4: On Board Single Seven Segment.
5a: Switch to Reset Microcontroller.
5b: Switch to Simulate Digital Input.
5c: Switch to Simulate Digital Input & Interrupt.
6: PWM (Pulse Width Modulation) Section.
7: On Board Buzzer Interface.
8: Transistor H-Bridge DC Motor Interface.
9: Unipolar Stepper Motor Interface Using ULN2003.
10: ADC (Analog to Digital Conversion) Section.
11: On Board 2x16 8-Bit LCD Interface Connector.
12: Rs232 Communication Section Using MAX232.
13: Serial External EEPROM Interface.
14: 4 Pin I/O Pin Connector.
15: Var Resistance for LCD Contrast Adjustment.
16: On Board 20Mhz Crystal Oscillator.
17: 4x2 PIN I/O Pin Connector.

ICSP Programming

The PICBox1 has an on-board ICSP programming connector, which allows you to establish a connection between the microcontroller and to ICSP programmer. Use ICSP programmer to load a Hex file into the microcontroller, below figure shows the connection between a compiler, PICpgm programmer and microcontroller.

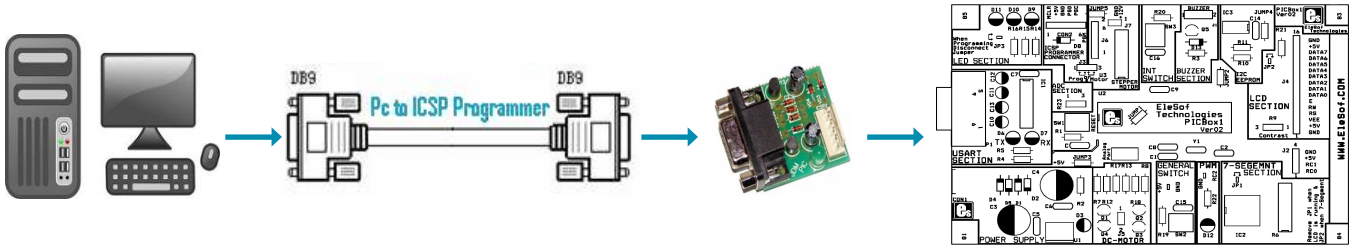


Write a code in some of PIC compilers, generate a.hex and through ICSP programmer, programm PIC controller on the board.



ICSP Programming Connector & Con3 Jumper Setting for Programming

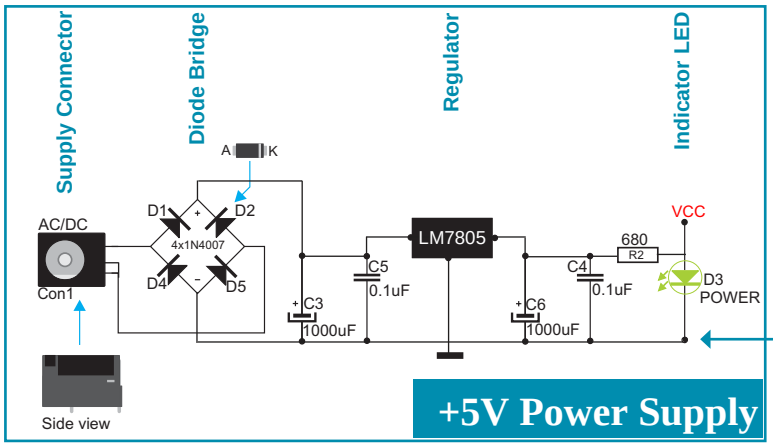
Connection For Programming On-Board Controller

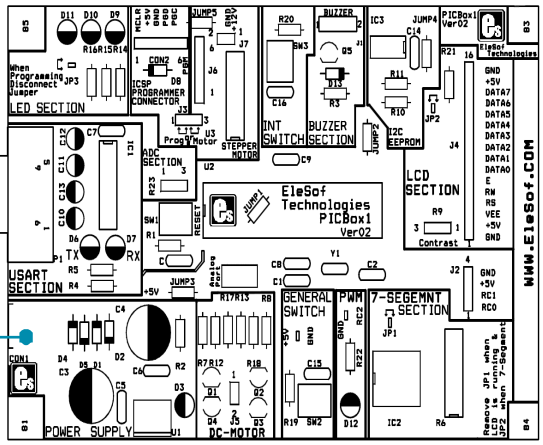


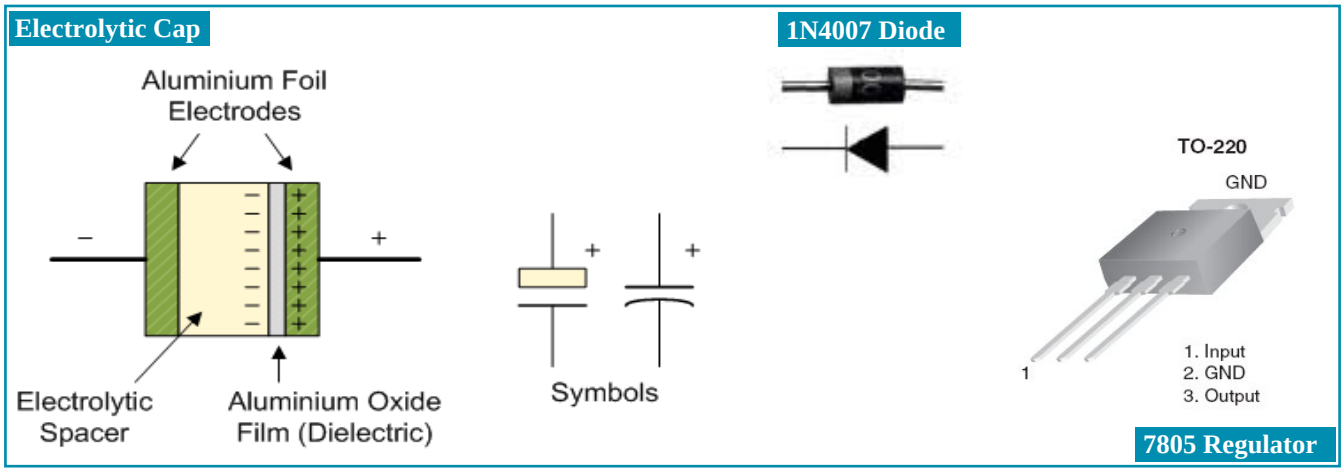
Note: For more information on the PICpgm and CCS Compiler refer to the relevant manual provided in the PICBox1 development system package & for burning hex to controller first checks that J3 Jumper '2' & '3' pins are connected. If during programming verification fail than plug the power supply to PICBox Development Board through adapater which is given with Kit & a.gain try for programming the Board.

Power Supply

The PICBox1 development system have on-board +5V regulated power supply, which has bridge rectifiers, 7805 regulator and filter circuit. With power supply one LED is connected which indicate that board is power ON. This board supports both DC & AC adaptor.







About Power Supply

This +5 volt power supply is based on the commercial 7805 voltage regulator IC. This IC contains all the circuitry needed to accept any input voltage from 8 to 18 volts and produce a steady +5 volt output, accurate to within 5% (0.25 volt). It also contains current-limiting circuitry and thermal overload protection, so IC won't be damaged in case of excessive load current; it will reduce its output voltage instead.

The 1000µf capacitor serves as a "reservoir" which maintains a reasonable input voltage to the 7805 throughout the entire cycle of the ac line voltage. The two rectifier diodes keep recharging the reservoir capacitor on alternate half-cycles of the line voltage, and the capacitor is quite capable of sustaining any reasonable load in between charging pulses. The 10µf and .01µf capacitors serve to help keep the power supply output voltage constant when load conditions change, the electrolytic capacitor smoothes out any long-term or low frequency variations. However, at high frequencies this capacitor is not very efficient. Therefore, the .01µf is included to bypass high-frequency changes, such as digital IC switching effects, to ground.

The LED and its series resistor serve as a pilot light to indicate when the power supply is on. I like to use a miniature LED here, so it will serve that function without being obtrusive or distracting while I'm performing an experiment. I also use this LED to tell me when the reservoir capacitor is completely discharged after power is turned off. Then I know it's safe to remove or install components for the next experiment.

LED's

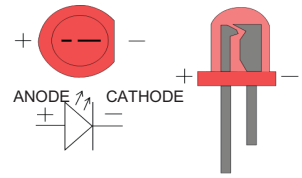
LED diode (Light-Emitting Diode) is a highly efficient electronic light source. When connecting LED's than it is necessary to place a current limiting resistor the value of that current limiting resistor by formula R=U/I where R referred to resistance expressed in ohms, U is referred to voltage on the LED and I stands for LED diode current. A common LED diode voltage is approximately 2.V, while the current varies from 1mA to 20mA depending on the type of LED diode.

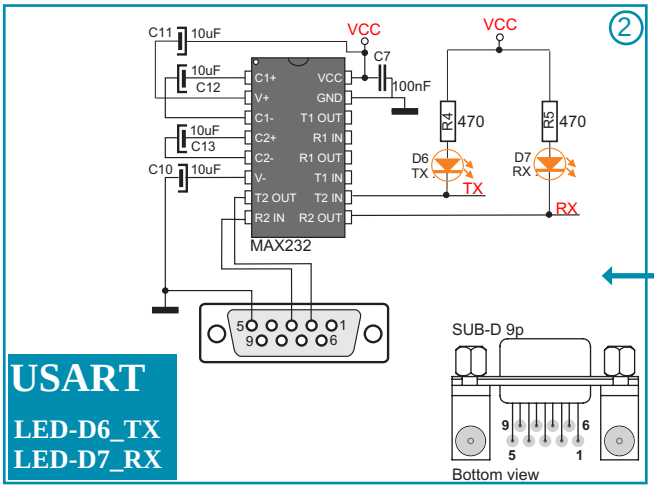
On-Board

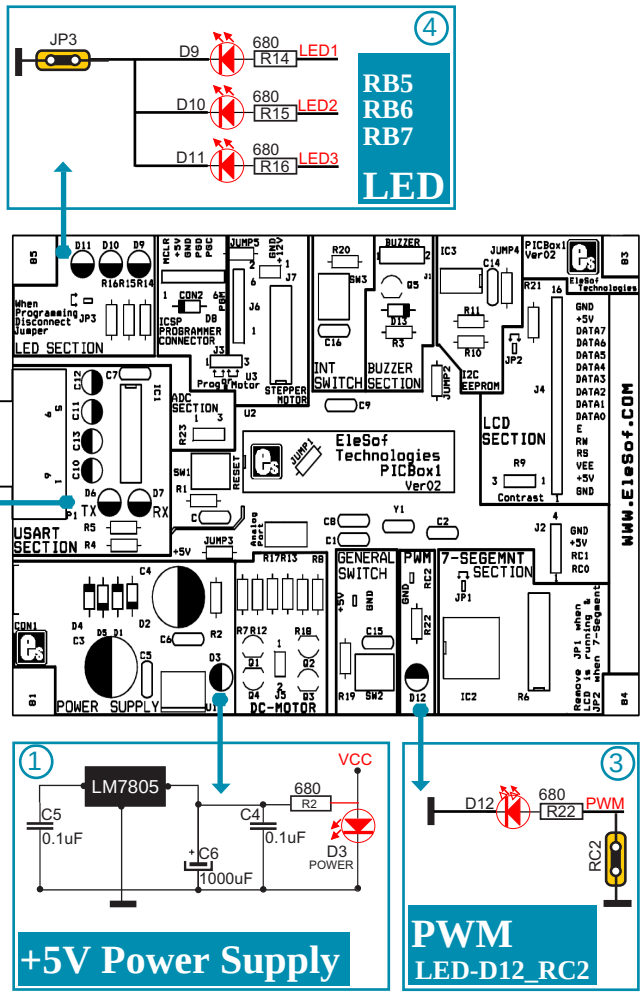
The PICBox1 has 7 LED's on-board; the function of each LED's is described below.

- LED D3 diode is power indicator LED, which shows the status of board weather it ON/OFF.
- LED's D6 & D7 diode are connected Rs232 communication section. The D6 LED diode shows data transmitting status by the Microcontroller & D7 LED diode shows data receiving status by the Microcontroller.
- LED D12 diode is connected with RC2/CCP1 pin of Port 'C', this LED can use as general purpose LED to read the status of pin High(1) or Low(0). It can also be used as, to implement PWM (Pulse Width Modulation) on-board.
- LED D9, D10 & D11 are connected with RB5, RB6 & RB7 pins of Port 'B' respectively, these LED's can be used to perform general function or to design tracker.

LED Diode







Note: To operate LED's check that jumper JP3 is connected.

Seven Segment

The seven-segment LED display has four individual digits, each with a decimal point. Each of the seven segments (and the decimal point) in a given digit contains an individual LED. When a suitable voltage is applied to a given segment LED, current flows through and illuminates that segment LED. By choosing which segments to illuminate, any of the nine digits can be shown. For example, as shown in the figure below, a 2 can be displayed by illuminating segments a, b, d, e, and g.

Seven segment displays can be divided into 2 types of connection. One is called common anode of which all the anodes of the LED's are connected together, leaving the cathodes open for connection. The other one is called common cathode of which all the cathodes of the LED's are connected together, leaving the anodes open for connection from microcontroller side.

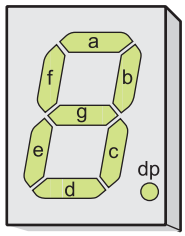
It is necessary to place a current limiting resistor the value of that current limiting resistor by formula R=U/I where R referred to resistance expressed in ohms, U is referred to voltage on the LED and I stands for LED diode current. A common LED diode voltage is approximately 2.V, while the current varies from 1mA to 20mA depending on the type of LED diode.

On-Board

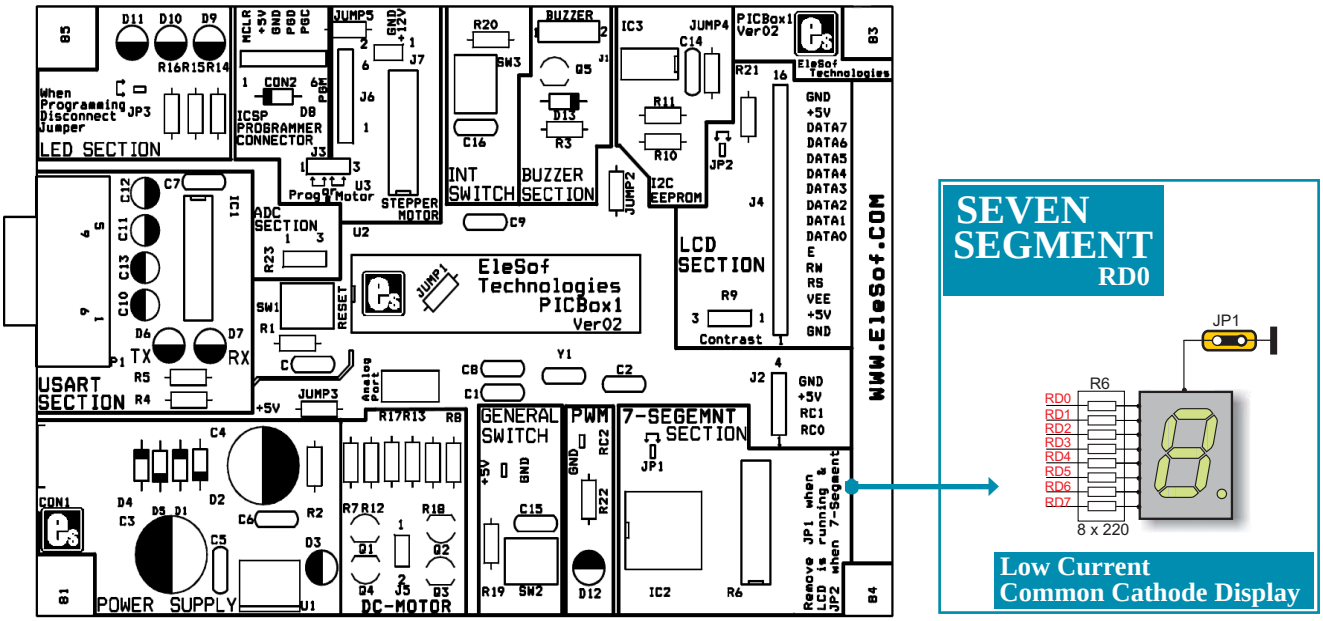
The PICBox1 has 1 on-board common cathode Seven Segment. The Seven Segment is interface with Port 'D' of microcontroller, the details of connection mentioned below.

7 Segment Pin interface details.

Segment - a	RD5 pin, Port 'D'
Segment - b	RD4 pin, Port 'D'
Segment - c	RD2 pin, Port 'D'
Segment - d	RD1 pin, Port 'D'
Segment - e	RD0 pin, Port 'D'
Segment - f	RD6 pin, Port 'D'
Segment - g	RD7 pin, Port 'D'
Segment - dp	RD3 pin, Port 'D'



Common marking of
7-segment display segments

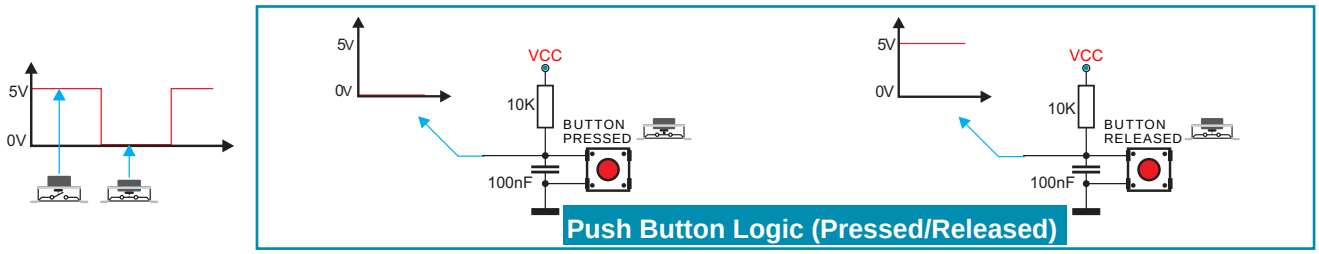


Note: To operate Seven Segment check that jumper JP1 is connected & 16x2 LCD is not connected with connector J4, because LCD & Seven Segment both using same data port (Port 'D').

Switch's

The Logic state of all microcontroller digital inputs may be changed using push buttons or switchs.

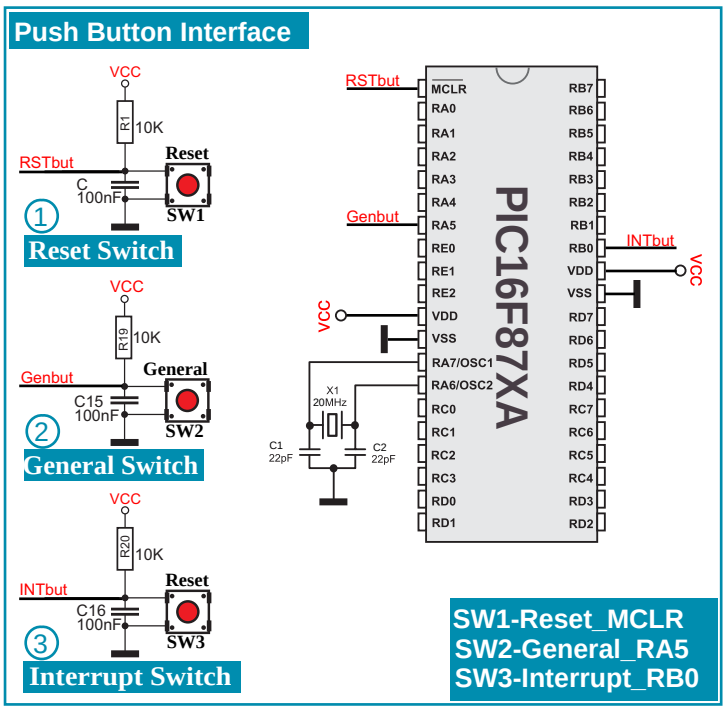
Here we can se that a switch is interface with microcontroller port with pull-up resistor for limit current and with capacitor parallel to switch for protecting microcontroller from spikes which can be generated at the time of switch pressing.

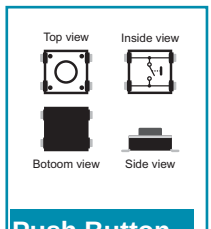


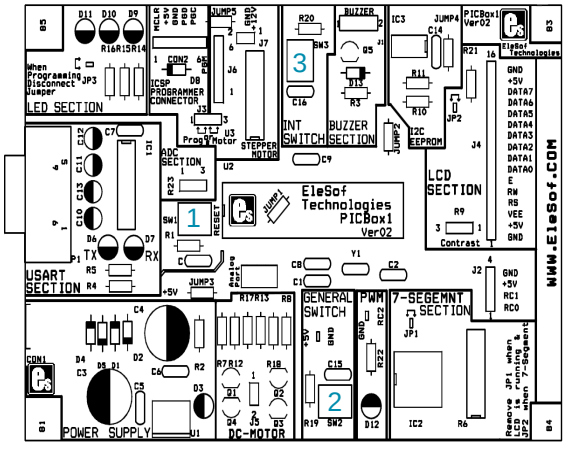
On-Board

This PICBox1 having three switchs, their details are.

- Sw1 switch is connected with MCLR pin of microcontroller, this switch work as Reset switch (Reset Microcontroller).
- Sw2 switch is connected with RA5 pin of Port 'A' of microcontroller, this switch can be used as changing the status of controller pin in General mode only.
- Sw3 switch is connected with RB0 pin of Port 'B' of microcontroller, this switch used as changing the status of controller pin in General mode & Interrupt mode.







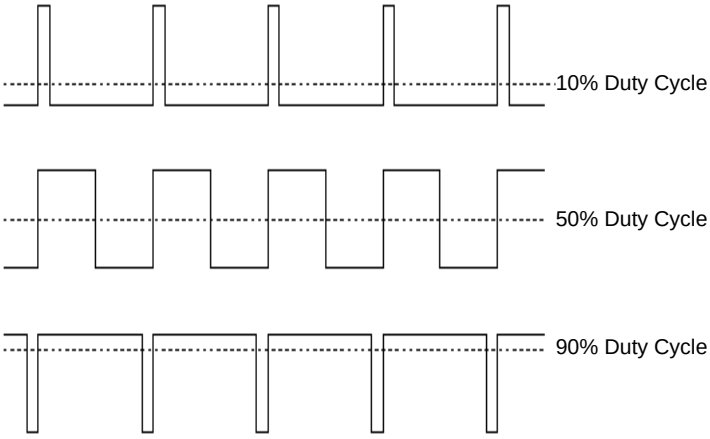
PWM

PWM (Pulse Width Modulation) is the term used to describe using a digital signal to generate an analogue Output signal. This is usually used to control the average power to a load in a motor speed control circuit.

You can also use it to generate a continuously variable analogue output without using any other integrated Circuits by smoothing the PWM signal using a capacitor.

As well as saving the costs of extra chips and interfaces the Pulse Width Modulation signal will not drift over time since it is generated from the time base of the processor i.e. a quartz crystal. Using analogue circuits to generate accurate signals that don't drift is a difficult task so PWM is very effective and cheap. It works by changing the average voltage level and this is done by generating a constant frequency signal but one where the pulse width is changed (or modulated).

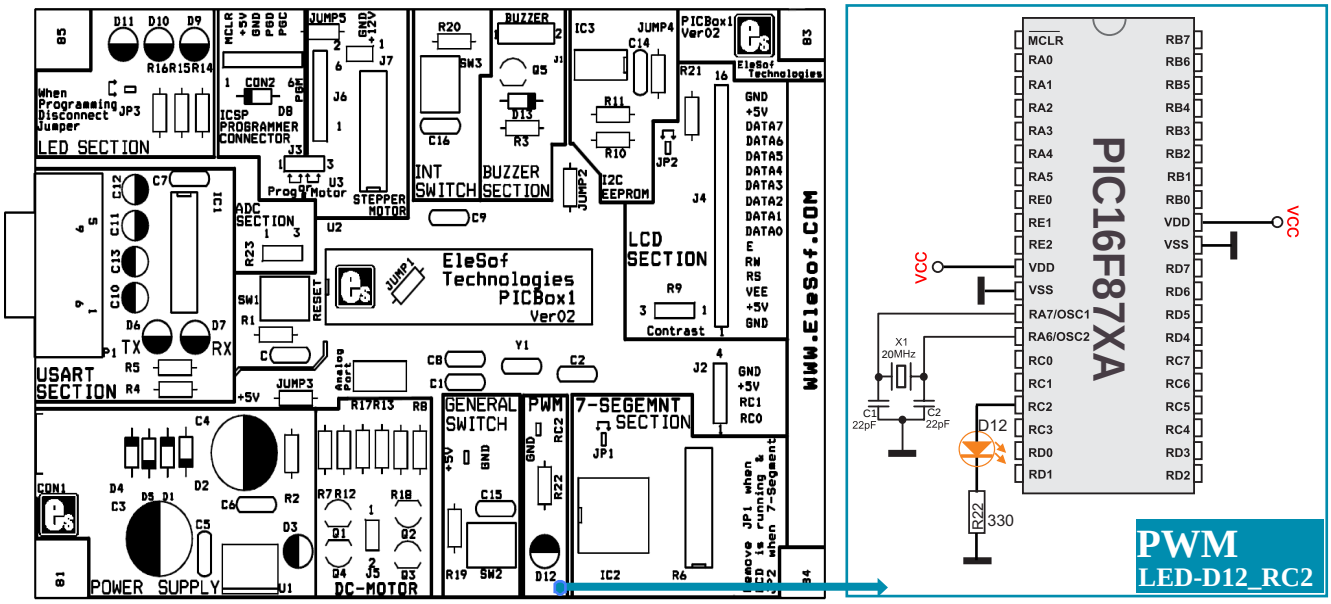
For a moment if you think of the digital signal when it is at its extremes i.e. normal - it generates the maximum of 5V when the output is high and the minimum of 0V when the output is low. If you want to generate a 2.5V signal then you need to make the signal on for half of the time and off for the rest and then take the average.



In the diagram the digital signal (solid line) is at a constant frequency while the pulse width is changed (modulated). The dotted line represents the average signal (if the digital signal is converted to an average). The duty cycle represents the amount of time that the signal is high compared to the amount of time that the signal is low.

On-Board

For implementing PWM function (LED Dimming) over PICBox1, there is one LED diode D12 is connected with RC2/CCP1 pin of Port 'C', with current limitation resistor R22 in series.

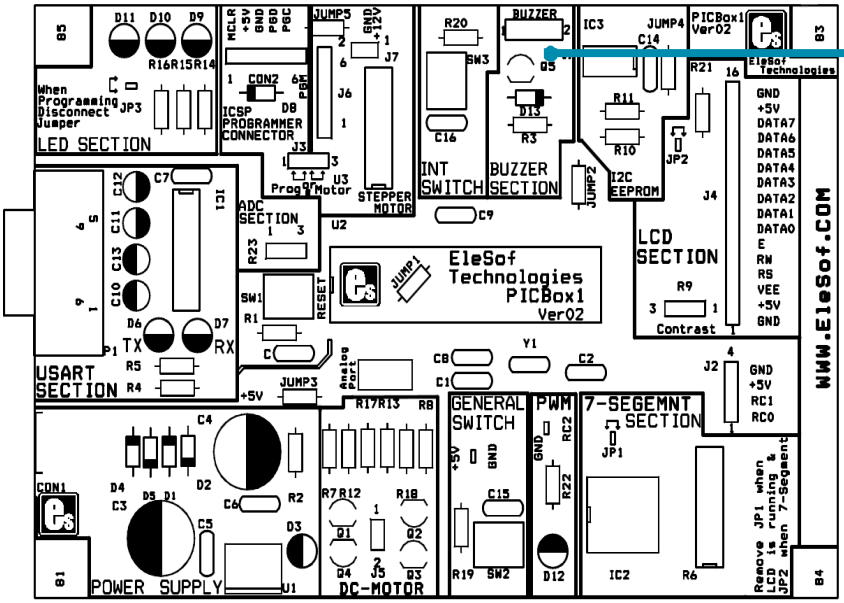


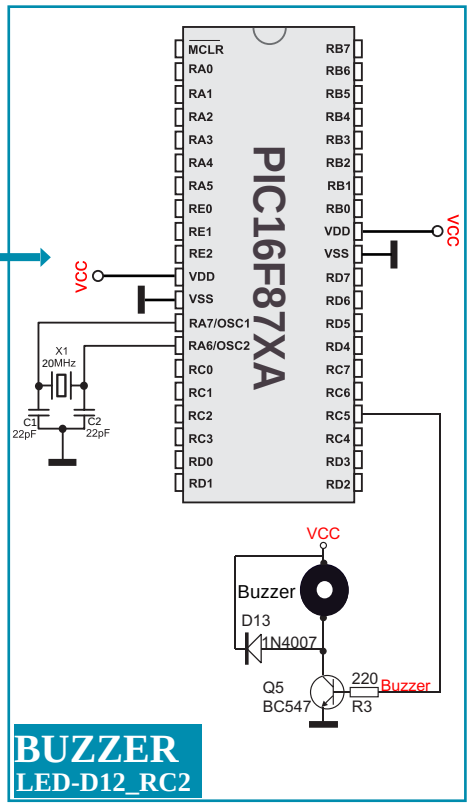
Buzzer

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or electronic. Typical uses of buzzers and beepers include alarms, timers and confirmation of user input such as a Switch click or any other event occurs.

On-Board

PICBox1 have one electronic buzzer interface with microcontroller by the help of transistor Q5 working on switching mode, it have current limiting resistor through base and pin RC5 of Port 'C'. There is also free wheeling diode D13 1N4007 which is connected parallel with Buzzer in reverse biased, to protect the transistor Q5 from reverse current if generated.





Here are some of the primary benefits of using electric buzzers:

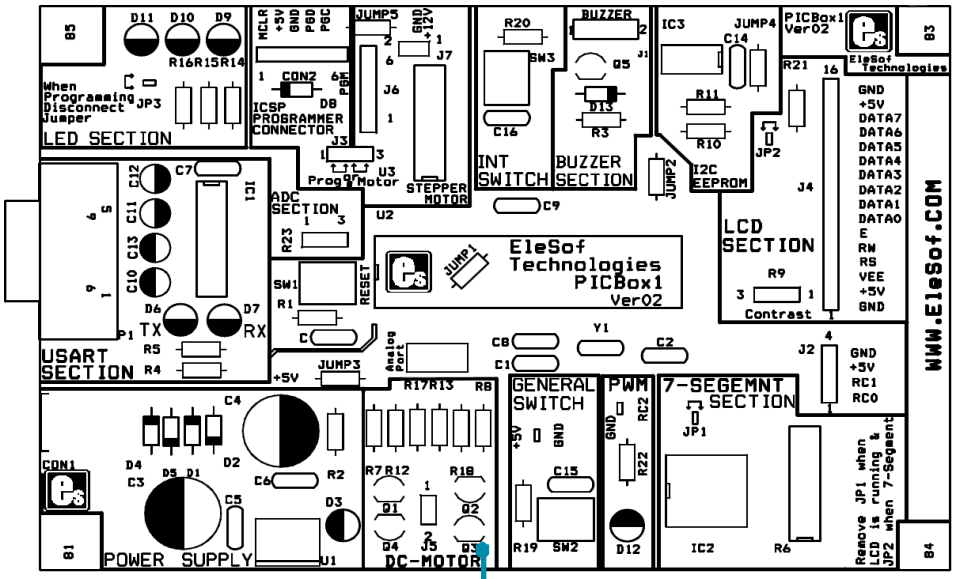
- The use of warning systems like delta-alarm or electric buzzers could be very beneficial in minimizing loss of lives during a disaster or accident. They are important devices in any building or facilities to alert and notify people if a timely evacuation is necessary.
- Specialized electric alarm systems could act as warning about threatening liquid level conditions in lift pump chambers, sewage, and other non-potable water applications. It is essential that you know how efficiently your home is or business site's sewer system works and this can be gauged more accurately if it is accompanied by an electric sewage alarm.
- Most electro mechanical buzzers are easy to set up. In fact, you don't need to hire an electrician to install it, since no hard wiring is usually needed. This means cutting down on expenditure for hiring a professional installer.
- In workplace, electric buzzers, especially those with timing software, offers more benefits and features than traditional timers and expensive bell. They could be synchronized with automatic software via computer to control and switch times or channels within the day.
- Electric buzzers could even be used in a wide array of appliances. Some buzzers are now used in switching sounds of electric home appliances such as microwaves, washing machines, calculators, smoke detectors, and telephone ringers, transmitters, and many more.
- With the help of electro-mechanical buzzers, you can be notified of an automobile entering your residence or business when you're in remote areas, so you never miss a warning.

DC Motor

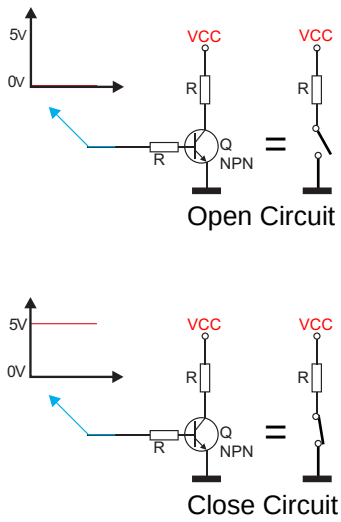
DC motors are fairly simple to understand. They are also simple to make and only require a battery or dc supply to make them run. We can say in simple words that an electro mechanical system works when direct current is supplied to it.

On-Board

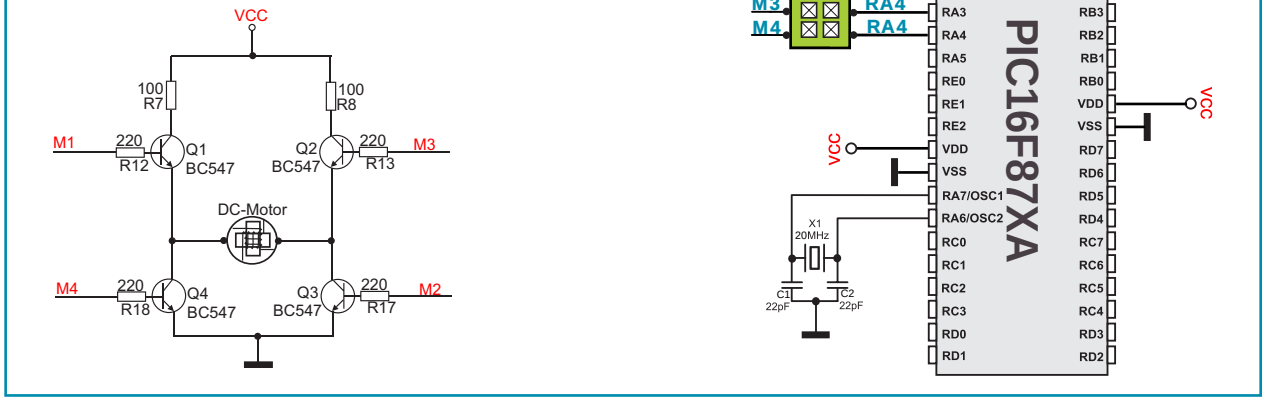
Here motor is interface with controller by using An H-bridge transistors arrangement to control speed & direction of motor. Here transistors used are operating in switching mode, the transistor used in this H-bridge are BC547 (Q1, Q2, Q3 & Q4) with current limiting resistor at their base (R12, R13, R17 & R18), connected with (RA1, RA3, RA2 & RA4) of Port 'A'.



Transistor as Switch



H-Bridge DC MOTOR



Note: RA4 is a Open Drain when configure as Output, so when configure as output pullup this with 10K resistor .

Stepper Motor

A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. The motors rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of motor shafts rotation. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied. There are three basic stepper motor types they are:

- Variable-reluctance
- Permanent-magnet
- Hybrid

On-Board

There is 'J6' 5pin connector for connecting permanent-magnet stepper motor with Port 'B' of microcontroller by using ULN2003 (U3) IC & there is 'J7' on board 2pin connector for providing external +12V supply to ULN2003 & Stepper motor.

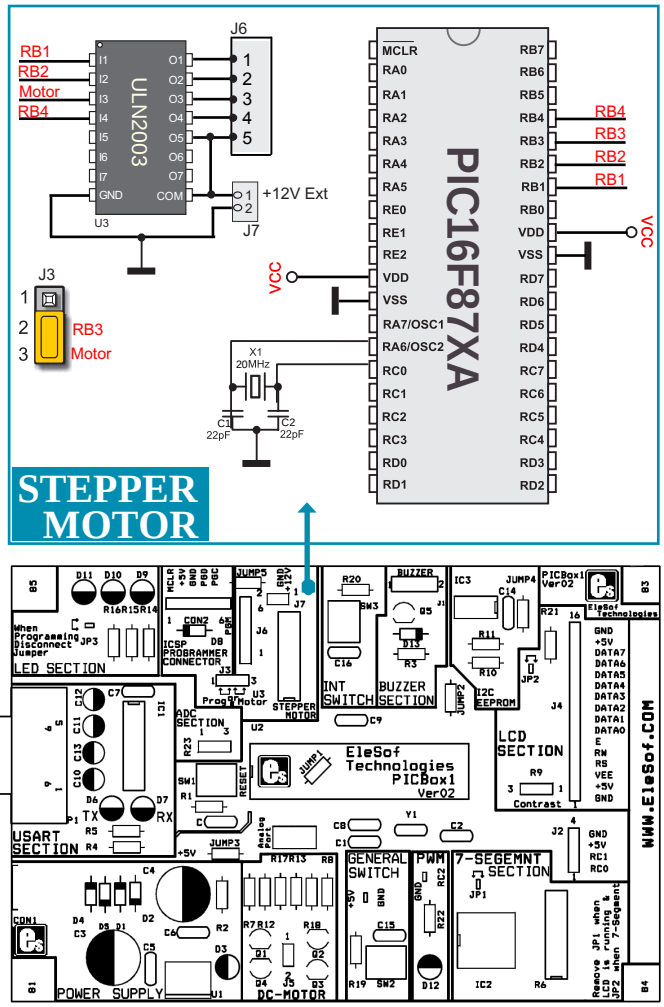
The ULN2003 internally employs high voltage, high current Darlington arrays each containing seven open collector Darlington pairs with common emitters. ULN2003 used as buffer to drive because stepper motor, because it need high current which is not able to provide that much current.

Stepper Motor Advantages.

- The rotation angle of the motor is proportional to the input pulse.
- The motor has full torque at stand still (if the windings are energized)
- Precise positioning and repeat ability of movement since good stepper motors have an accuracy of 3–5% of a step and this error is non cumulative from one step to the next.
- Excellent response to start/stop/reverse.
- Very reliable since there are no contact brushes in the motor. Therefore the life of the motor is simply dependant on the life of the bearing.
- The motors response to digital input pulses provides open-loop control, making the motor simpler and less costly to control.
- It is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft.
- A wide range of rotational speeds can be realized as the speed is proportional to the frequency of the input pulses.

Stepper Motor Disadvantages.

- Resonances can occur if not properly controlled.
- Not easy to operate at extremely high speeds.

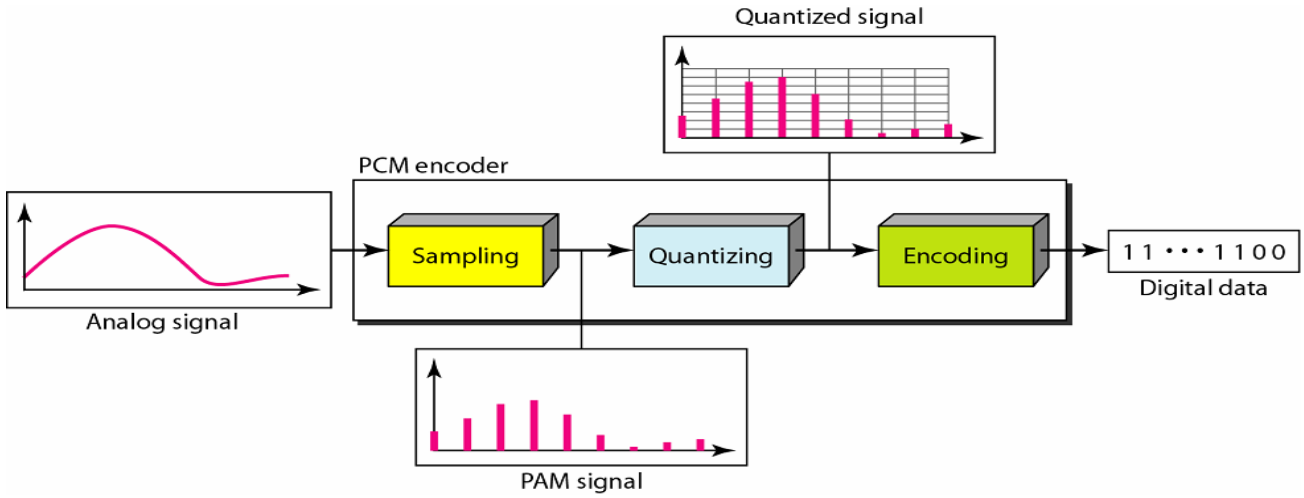


Note: To operate Stepper Motor first checks that J3 Jumper '1' & '2' pins are connected.

ADC

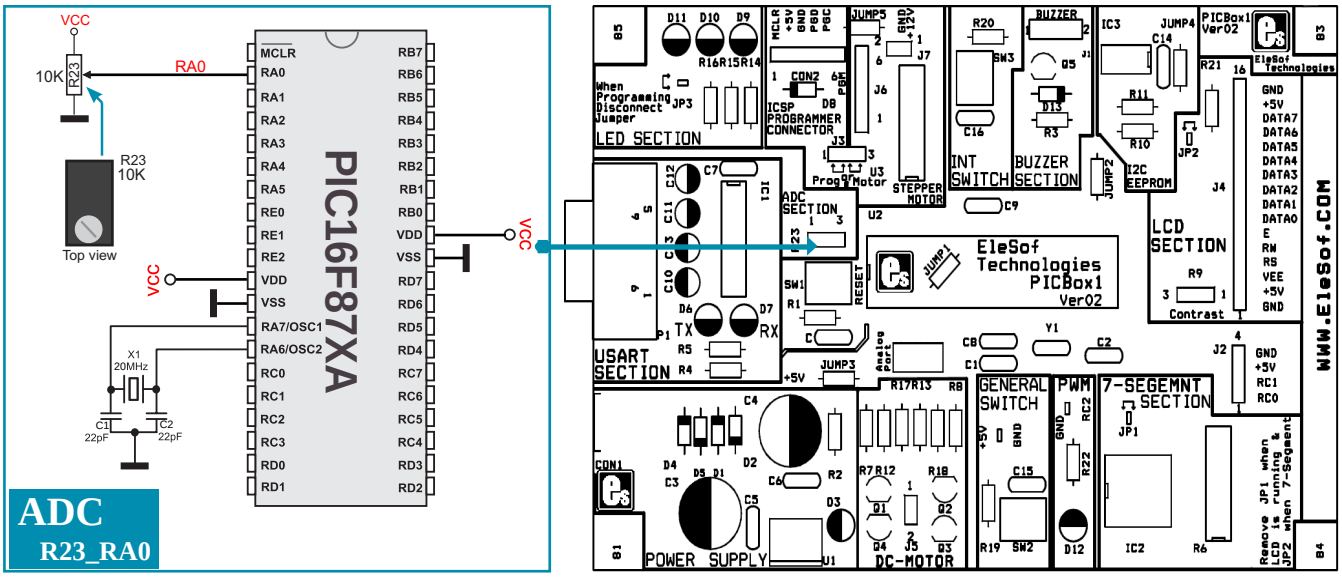
An A/D converter is used for the purpose of converting an analog signal into appropriate digital value. A/D converter is linear, which means that the converted number is linearly dependent on the input voltage value.
The A/D converter built into the microcontroller provided with PICBox1 development system converts an analog voltage value into a binary number. Voltages varying from 0V to 5V DC may be supplied through the A/D test input. Basically, you can measure any analog signal that fits in the range acceptable by PIC (0-VCC).

Analog to Digital conversion process



On-Board

The value of the input analog voltage can be changed linearly using potentiometer R23 of 10K ohm, which is connected with RA0 pin of Port 'A' of microcontroller.



Application of ADC (Analog to Digital Conversion)

- To measure the voltage.
- Reading the sensor's output which is in analog form (Temperature Sensor, Pressure Sensor, LPG Sensor, etc.).
- Interfacing touch screen.
- Interfacing multi switch's with single pin.

16x2LCD Display

The PICBox1 development system provides an on-board connector to plug alphanumeric 2x16 LCD display into. Such connector is connected to the microcontroller. Communication between an LCD display and the microcontroller is established using an 8-bit mode. Alphanumeric digits are display in two lines each containing up to 16 characters of 7x5 pixels.

Potentiometer or say variable resistor R9 is used for LCD contrast adjustment & jumper JP2 control the back light of display. If jumper is their, than backlight ON but when no than backlight remains OFF.

Commonly used LCD commands code:

COMMAND	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	EXECUTION TIME
Clear Display	0	0	0	0	0	0	0	0	0	1	1.64ms
Cursor Home	0	0	0	0	0	0	0	0	1	x	1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	40us
Display on/off control	0	0	0	0	0	0	1	D	U	B	40us
Cursor Display Shift	0	0	0	0	0	1	D/C	R/L	x	x	40us
Function Set	0	0	0	0	1	DL	N	F	x	x	40us
Set CGRAM Address	0	0	0	1	CGRAM Address						40us
Set DDRAM Address	0	0	1	DDRAM Address							40us
Read "BUSY" Flag (BF)	0	1	BF	DDRAM Address							----
Write to CGRAM or DDRAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	40us
Read from CGRAM or DDRAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	40us

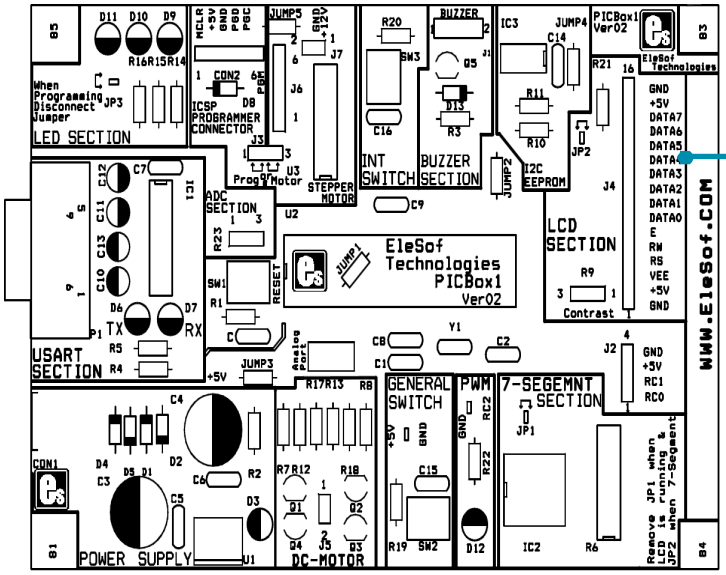
LCD Basic Commands

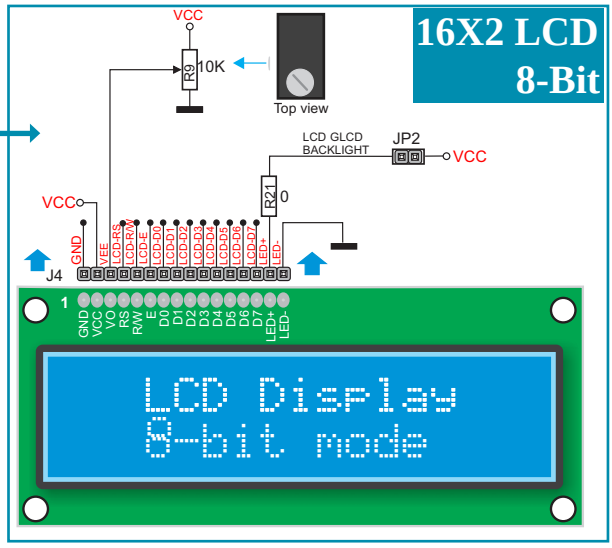
All data transferred to an LCD through the outputs D0-D7 will be interpreted as a command or a data, which depends on the RS pin logic state:

- RS = 1 - Bits D0 - D7 are addresses of the characters to be displayed. LCD processor addresses one character from the character map and displays it. The DDRAM address specifies location on which the character is to be displayed. This address is defined prior to transferring character or the address of the previously transferred character is automatically incremented.
- RS = 0 - Bits D0 - D7 are commands for setting the display mode.

LCD Pin interface details.

LCD Pin1-GND	GND
LCD Pin2-VCC	VCC +5V
LCD Pin3-VE	R9 Resistor for contrast
LCD Pin4-RS	RE0 pin, Port 'E'
LCD Pin5-R/W	RE1 pin, Port 'E'
LCD Pin6-E	RE2 pin, Port 'E'
LCD Pin7-D0	RD0 pin, Port 'D'
LCD Pin8-D1	RD1 pin, Port 'D'
LCD Pin9-D2	RD2 pin, Port 'D'
LCD Pin10-D3	RD3 pin, Port 'D'
LCD Pin11-D4	RD4 pin, Port 'D'
LCD Pin12-D5	RD5 pin, Port 'D'
LCD Pin13-D6	RD6 pin, Port 'D'
LCD Pin14-D7	RD7 pin, Port 'D'
LCD Pin15-LED+	JP2
LCD Pin16-LED-	GND

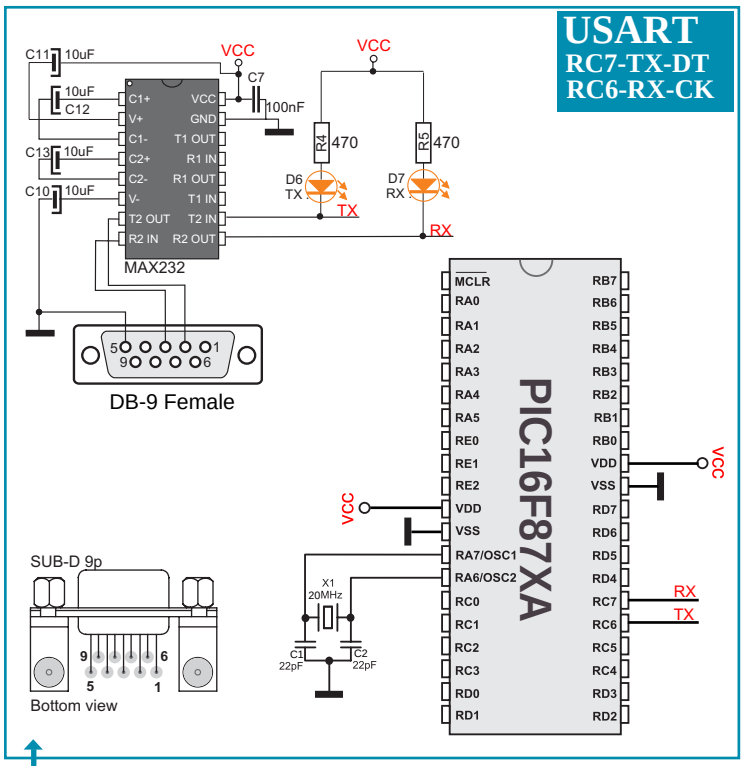


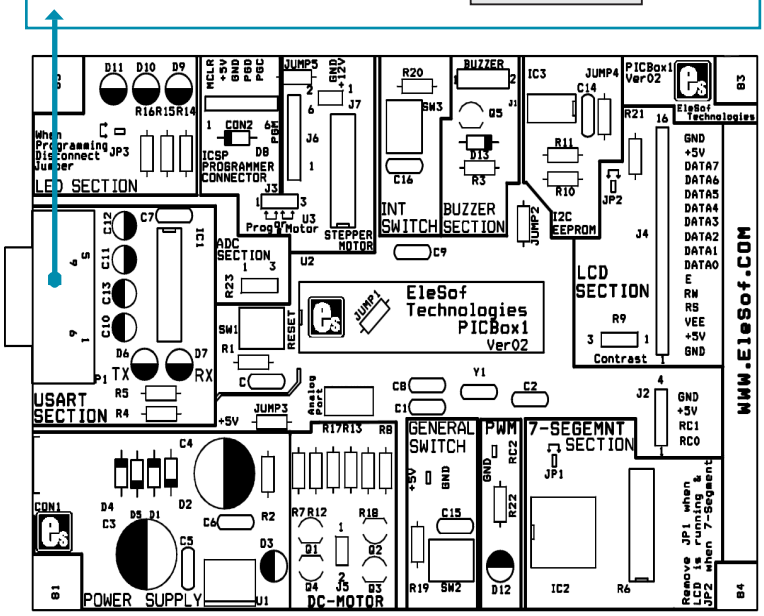


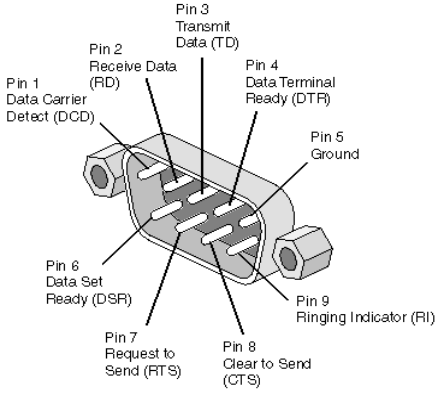
Rs232

RS-232 serial communication is performed through a 9-pin SUB-D connector and the microcontroller USART module. In order to enable such communication, it is necessary to establish a connection between RX and TX communication lines of microcontroller and MAX232 IC and RX. The microcontroller pins used in such communication are marked as follows: RX- receive data, TX- transmit data. Baud rate goes up to 115kbps.

The USART (universal synchronous/asynchronous receiver/transmitter) is one of the most common ways of exchanging data between the PC and peripheral components. In order to enable the USART module of the microcontroller to receive input signals with different voltage levels, it is necessary to provide a voltage level converter such as MAX-232.






Pin 1 Data Carrier Detect (DCD)
Pin 2 Receive Data (RD)
Pin 3 Transmit Data (TD)
Pin 4 Data Terminal Ready (DTR)
Pin 5 Ground
Pin 6 Data Set Ready (DSR)
Pin 7 Request to Send (RTS)
Pin 8 Clear to Send (CTS)
Pin 9 Ringing Indicator (RI)

About MAX-232

The **MAX232** is an integrated circuit that converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits. The MAX232 is a dual driver/receiver and typically converts the RX, TX, CTS and RTS signals.

The drivers provide RS-232 voltage level outputs (approx. ± 7.5 V) from a single + 5 V supply via on-chip charge pumps and external capacitors. This makes it useful for implementing RS-232 in devices that otherwise do not need any voltages outside the 0 V to + 5 V range, as power supply design does not need to be made more complicated just for driving the RS-232 in this case. The receivers reduce RS-232 inputs (which may be as high as ± 25 V), to standard 5 V TTL levels. These receivers have a typical threshold of 1.3 V, and a typical hysteresis of 0.5 V.

The later MAX232A is backwards compatible with the original MAX232 but may operate at higher baud rates and can use smaller external capacitors – 0.1 µF in place of the 1.0 µF capacitors used with the original device. The newer MAX3232 is also backwards compatible, but operates at a broader voltage range, from 3 to 5.5V.

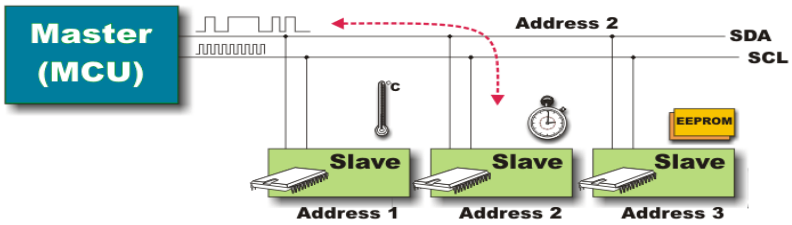
I²C EEPROM

EEPROM (Electrically Erasable Programmable Read-Only Memory) is a built-in memory module used to store data that should be saved when the power supply goes off. The 24CXXX circuit can store up to 1Kbit data and communicates with the microcontroller through SCL (Serial Clock) & SDA (Serial Data) pins of EEPROM IC. Data & Clock line is pull-up with resistor R10 & R11. The value of resistor defines the speed of I2C communication with Microcontroller.

About I²C Mode

I²C mode (*Inter IC Bus*) is especially suitable when the microcontroller and an integrated circuit (memories, temperature sensors, real-time clocks etc.). Similar to serial communication in SPI mode, data transfer in I²C mode is synchronous and bidirectional. This time only two pins are used for data transmission. These are the SDA (Serial Data) and SCL (Serial Clock) pins. The user must configure these pins as inputs or outputs through the TRISC bits.

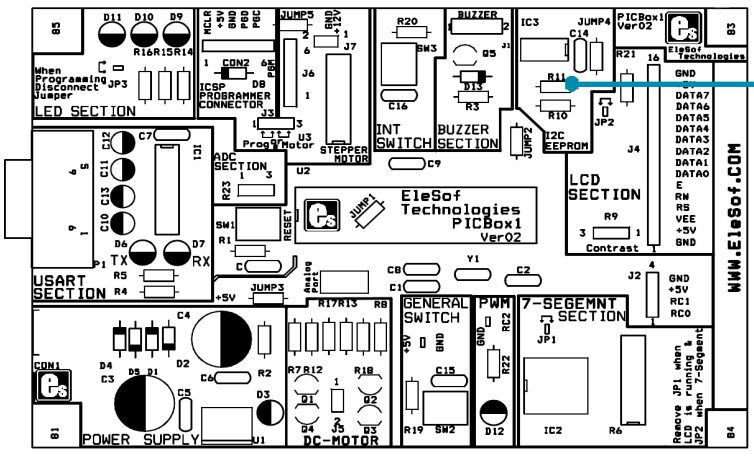
By observing particular rules (protocols), this mode enables up to 122 different components to be simultaneously connected in a simple way by using only two valuable I/O pins. *Clock*, necessary to synchronize the operation of both devices, is always generated by a master device (a microcontroller) and its frequency directly affects the baud rate.

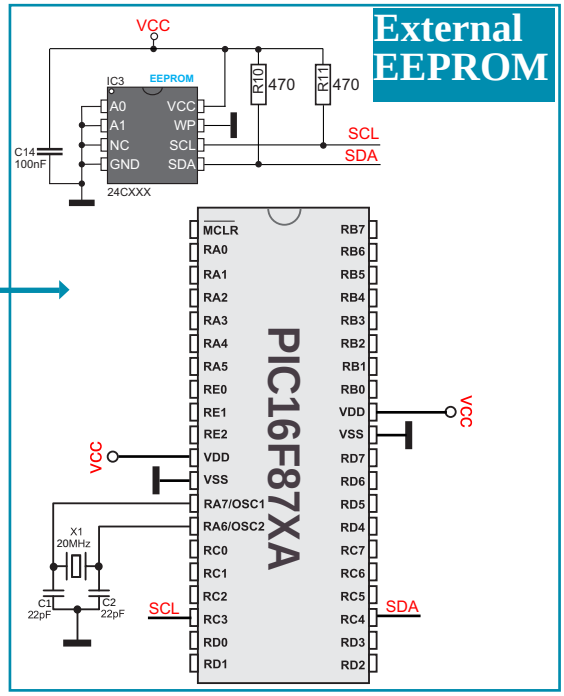


When *master* and *slave* components are synchronized by the clock, every data exchange is always initiated by the *master*. The master device first sends the START bit (logic zero) through the SDA pin, then a 7-bit address of the selected slave device, and finally, the bit which requires data write (0) or read (1) to the device. All *slave* devices sharing the same transmission line will simultaneously receive the first byte, but only one of them has the address to match and receives the whole data. Once the first byte has been sent (only 8-bit data are transmitted), *master* goes into receive mode and waits for acknowledgment from the receive device that address match has occurred. If the *slave* device sends acknowledge data bit (1), data transfer will be continued until the *master* device (microcontroller) sends the Stop bit.

On-Board

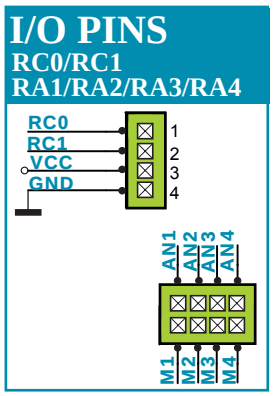
- EEPROM 'SDA' pin connected with RC4/SDI/SDA pin of controller, with pull-up resistor R10.
- EEPROM 'SCL' pin connected with RC3/SCK/SCL pin of controller, with pull-up resistor R11.





I/O Pins

Along with multifunction provided in PICBox1, there is four pin connector J2. The first two pins of connector are connected with RCO/T1OSO/T1CKI & RC1/T1OSI/CCP2 pins of Port 'C' respectively, & rest two pins of connector connected with +5V VCC and Ground.



Pin RC0/T1OSO/T1CKI used as:
- Digital input/output.
- Timer1 oscillator output.
- Timer1 external clock input.

Pin RC1/T1OSI/CCP2 used as:
- Digital input/output.
- Timer1 oscillator input.
- Capture2 input, Compare2 output, PWM2 output.

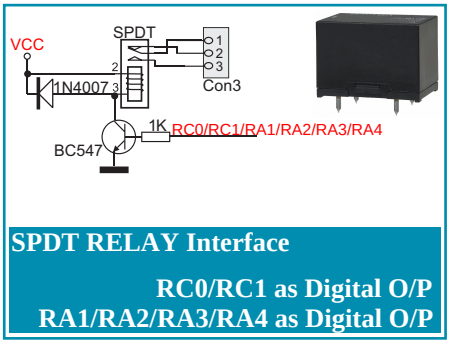
Pin RA1/AN1 used as:
- Digital input/output.
- Analog input1.

Pin RA2/AN2 used as:
- Digital input/output.
- Analog input2.
- Negative Analog voltage reference.

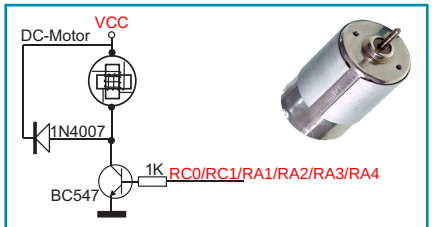
Pin RA4/T0CKI used as:
- Digital input/output.
- Clock input for Timer0.

Pin RA3/AN3 used as:
- Digital input/output.
- Analog input3.
- Positive Analog voltage reference.

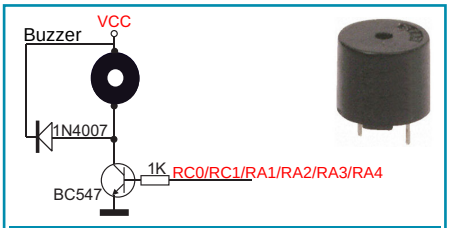
Here few examples interface circuit given below to interface external devices for implementing a function by using pins as:
- As Digital input/output:
- Timer1 external clock input:
- PWM2 output.
- ADC.



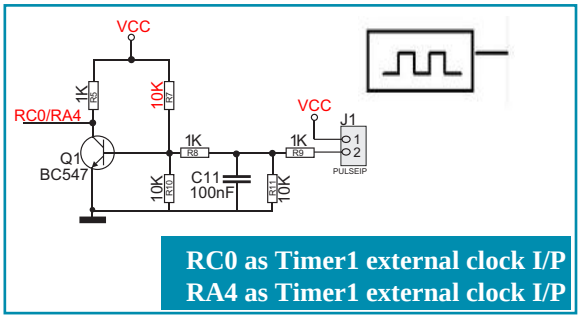
SPDT RELAY Interface
RC0/RC1 as Digital O/P
RA1/RA2/RA3/RA4 as Digital O/P



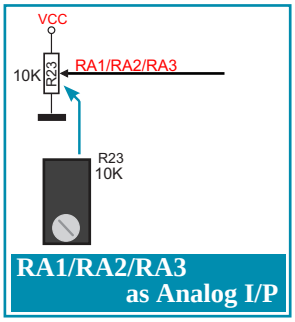
DC MOTOR Interface
RC0/RC1 as Digital O/P
RA1/RA2/RA3/RA4 as Digital O/P



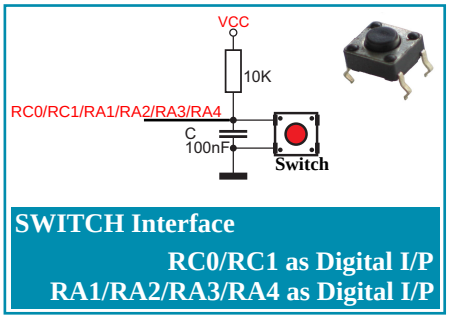
BUZZER Interface
RC0/RC1 as Digital O/P
RA1/RA2/RA3/RA4 as Digital O/P



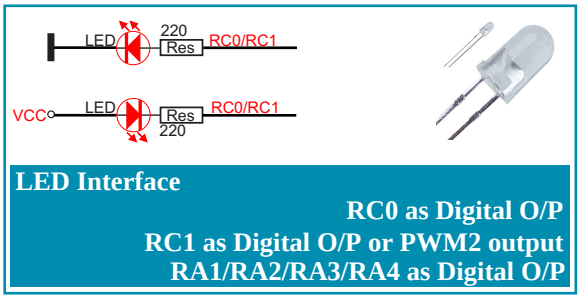
RC0 as Timer1 external clock I/P
RA4 as Timer1 external clock I/P



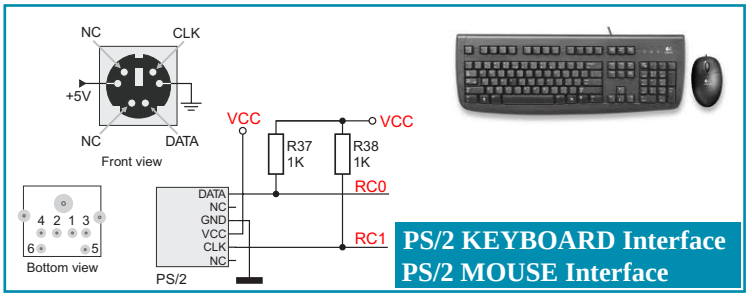
RA1/RA2/RA3 as Analog I/P



SWITCH Interface
RC0/RC1 as Digital I/P
RA1/RA2/RA3/RA4 as Digital I/P



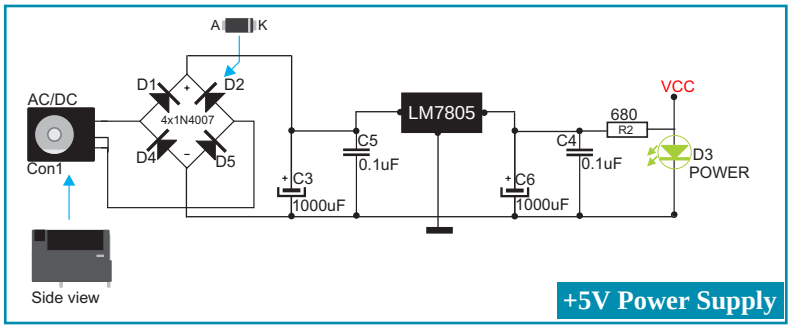
LED Interface
RC0 as Digital O/P
RC1 as Digital O/P or PWM2 output
RA1/RA2/RA3/RA4 as Digital O/P



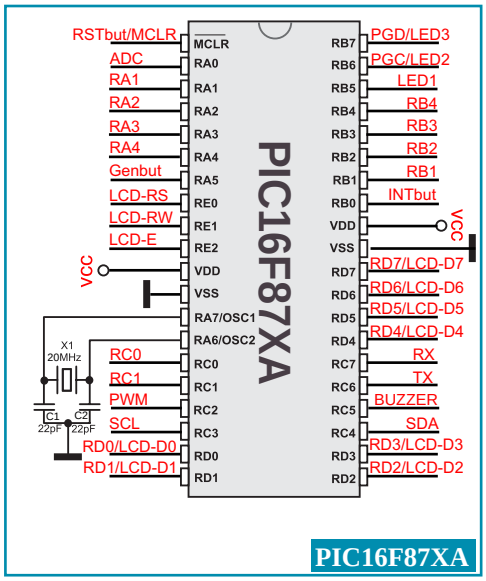
PS/2 KEYBOARD Interface
PS/2 MOUSE Interface

Note: RA4 is a Open Drain when configure as Output, so when configure as output pullup this with 10K resistor .

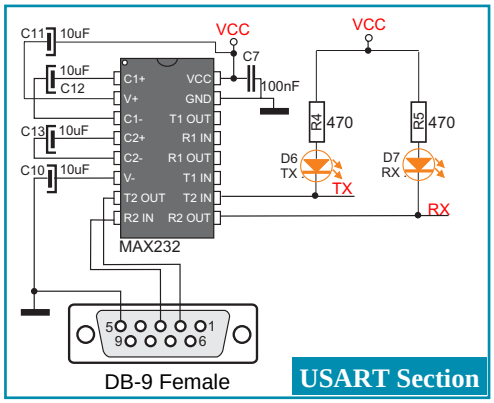
Schematic of PICBox1



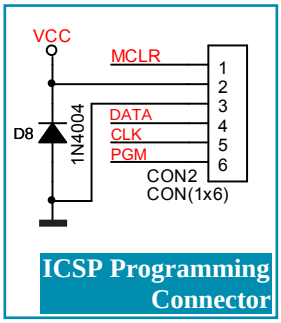
+5V Power Supply



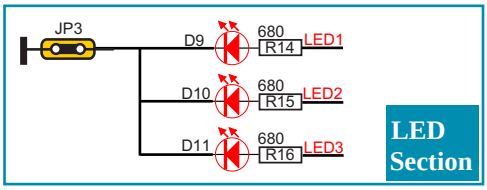
PIC16F87XA



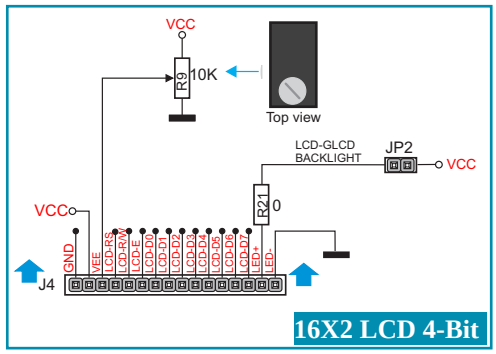
USART Section



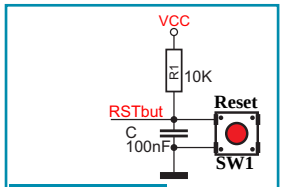
ICSP Programming Connector



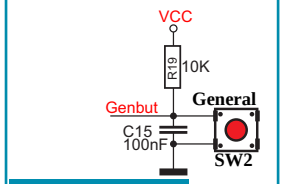
LED Section



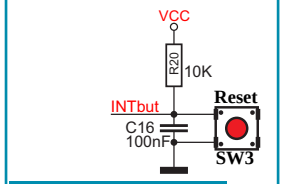
16X2 LCD 4-Bit



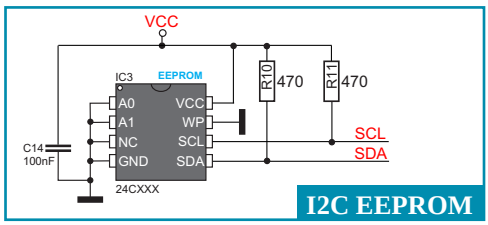
Reset Switch



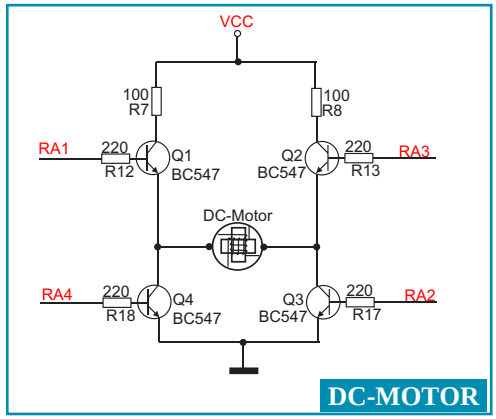
General Switch



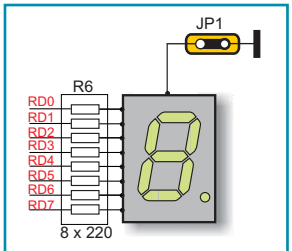
Interrupt Switch



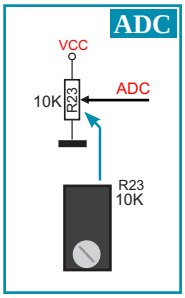
I2C EEPROM

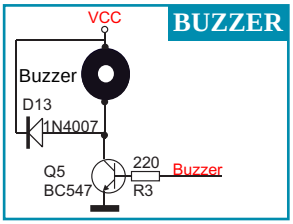


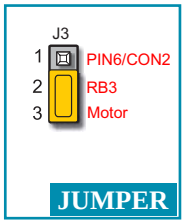
DC-MOTOR



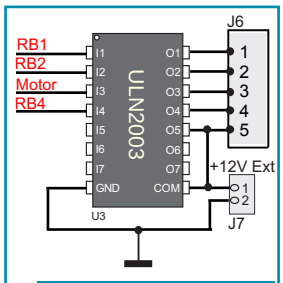
7-SEGMENT



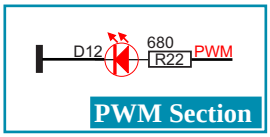




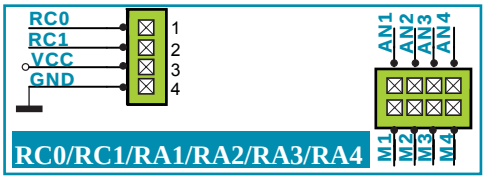
JUMPER



ULN2003 STEPPER MOTOR



PWM Section



RC0/RC1/RA1/RA2/RA3/RA4

Introduction to Embedded Systems

What is an Embedded System? The first question that needs to be asked is "What exactly is an embedded system?" To be fair, however, it is much easier to answer the question of what an embedded system is not, than to try and describe all the many things that an embedded system can be. An embedded system is a system that is implemented for a particular purpose. In contrast, an average PC computer usually serves a number of purposes: checking email, surfing the internet, listening to music, word processing, etc.. However, embedded systems usually only have a single task, or a very small number of related tasks that they are programmed to perform.

Every home has several examples of embedded system. Any appliance that has a digital clock, for instance, has a small embedded microcontroller that performs no other task than to display the clock. Modern cars have embedded computers onboard that control such things as ignition timing and anti-lock brakes using input from a number of different sensors. A little example of embedded system is calculator which you used for calculations.Embedded system rarely has a generic interface, however. Even if embedded systems have a keypad and an LCD display, they are rarely capable of using many different types of input or output. An example of an embedded system with I/O capability is a security alarm with an LCD status display, and a keypad for entering a password.

In general, an Embedded System:

- Is a system built to perform its duty, completely or partially independent of human intervention?
- Is specially designed to perform a few tasks in the most efficient way.
- Interacts with physical elements in our environment, viz. controlling and driving a motor, sensing temperature, etc.

An embedded system can be defined as a control system or computer system designed to perform a specific task. Common examples of embedded systems include MP3 players, navigation systems on aircraft and intruder alarm systems. An embedded system can also be defined as a single purpose computer.

Most embedded systems are time critical applications meaning that the embedded system is working in an environment where timing is very important: the results of an operation are only relevant if they take place in a specific time frame. An autopilot in an aircraft is a time critical embedded system.

Where is the use of Embedded systems?
The uses of embedded systems are virtually limitless, because every day new products are introduced to the market that utilizes embedded computers in novel ways. In recent years, hardware such as microprocessors, microcontrollers, and FPGA chips have become much cheaper. So when implementing a new form of control, it's wiser to just buy the generic chip and write your own custom software for it.

From an implementation viewpoint, there is a major difference between a computer and an embedded system. Embedded systems are often required to provide Real-Time response. A Real-Time system is defined as a system whose correctness depends on the timeliness of its response. Examples of such systems are flight control systems of an aircraft, sensor systems in nuclear reactors and power plants. For these systems, delay in response is a fatal error. A more relaxed version of Real-Time Systems is the one where timely response with small delays is acceptable. Example of such a system would be the Scheduling Display System on the railway platforms. In technical terminology, Real-Time Systems can be classified as:

- ***Hard Real-Time Systems*** - systems with severe confine on the timeliness of the response.
- ***Soft Real-Time Systems*** - systems which can bear small variations in response times.
- ***Hybrid Real-Time Systems*** - systems which exhibit both hard and soft confine on its performance.

Examples of embedded system are: , and etc.

Introduction to PIC and CCS Compiler

About PIC

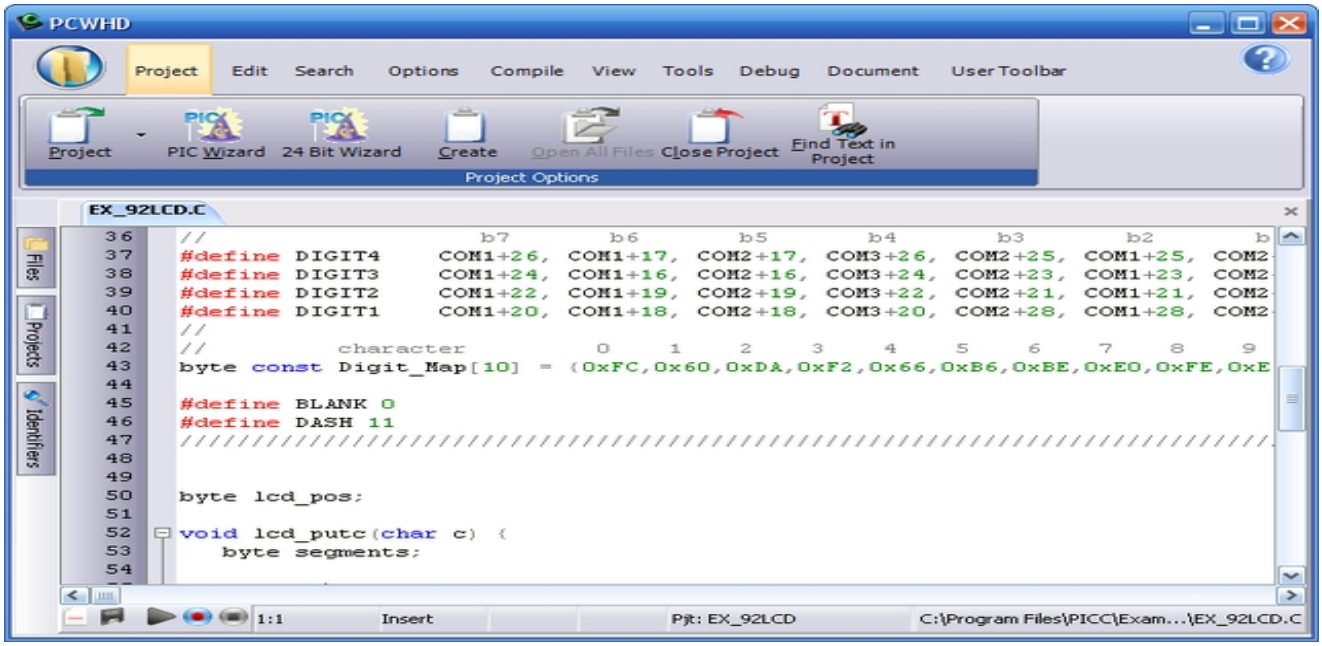
PIC is a family of RISC (Reduced Instruction Set Computer) microcontrollers made by Microchip Technology, derived from the PIC1650 originally developed by General Instrument's Microelectronics Division. Microchip Technology does not use PIC as an acronym; in fact the brand name is PIC micro.

What the name PIC stands for? Some think it is an acronym of "**P**eripheral **I**nterface **C**ontroller". However, I have been looking thru out the net and found that original General Instruments' acronym for the PIC1650 was "**P**rogrammable **I**ntelligent **C**omputer".

PICs are popular with developers and hobbyists alike due to their low cost, wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, and serial programming (and re-programming with flash memory) capability.

About CCS

CCS is embedded software tools designed for Microchip PIC and dsPIC. The CCS Compiler is comprised of Standard C operators and built-in libraries that are specific to PIC MCU registers, and access to hardware features from C. we can also say this is a cross compiler A **cross compiler** is a compiler capable of create executable code for a platform other than the one on which the compiler is run.

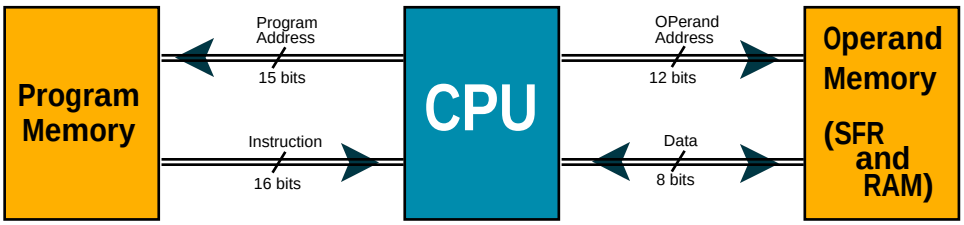


Architecture of PIC microcontroller

PIC is a family of Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1640.The **Harvard architecture** is computer architecture with physically separate storage and signal pathways for instructions and data. The term came from the **Harvard Mark I relay-based computer**, which stored instructions on **punched tape** and data in electro-mechanical counters. In Harvard architecture, there is no need to make the two memories share characteristics.

Punched tape or **paper tape** is a largely disused form of data storage, consisting of a long strip of paper in which holes are punched to store.

Harvard Architecture



Harvard Architecture

Feature of Harvard architecture

- Near all instructions are single instruction word instructions, (only one fetch per instruction).
- Instruction fetch and execute are pipelined so you can operate at near clock rate instructions per second.
- Separate buses one for instructions and one for data.

Limitations

The PIC architectures have several limitations:

- Only a single accumulator.
- A small instruction set.
- Operations and registers are not orthogonal; some instructions can address RAM and/or immediate constants, while others can only use the accumulator
- Memory must be directly referenced in arithmetic and logic operations, although indirect addressing is available via 2 additional registers
- Register-bank switching is required to access the entire RAM of many devices, making position-independent code complex and inefficient
- Conditional skip instructions are used instead of conditional branch instructions used by most other architectures.

Note: For more details over PIC & CCS refer to user manual of PIC & CCS given with PICBox1.

Overview of Embedded C

Programming in C Language

As the scale of microcomputer based systems has increased over the years, productivity and maintainability using Assembly language has become an issue. As a result, C language has become a popular alternative. The following explains the main features of the C language and describes how to write a program in "C".

Features of the C Language

1: An easily traceable program can be written.
The basics of structured programming, i.e., "sequential processing", "branch processing", and "repeat processing", can all be written in a control statement. For this reason, it is possible to write a program whose flow of processing can easily be traced.

2: A program can easily be divided into modules.
A program written in the C language consists of basic units called "functions". Since functions have their parameters highly independent of others, a program can easily be made into parts and can easily be reused. Furthermore, modules written in the assembly language can be incorporated into a C language program directly without modification.

3: An easily maintainable program can be written.
For reasons (1) and (2) above, the program after being put into operation can easily be maintained. Furthermore, since the C language is based on standard specifications (ANSI standard (Note)), a program written in the C language can be ported into other types of microcomputers after only a minor modification of the source program.

Program Development Procedure

The operation of translating a source program written in "C" into machine language is referred to as "compiling". The software provided for performing this operation is called a "compiler". Creation of a machine language file requires the conversion of start-up programs written in Assembly language and C language source files.

Program Rules

Since there is no specific format for C language programs, they can be written in any way desired as long as the stipulated rules of the C language are followed. However in order for a program to be easily read and maintained it should follow some common practices. This section explains some points for creating a well written program.

Rules on C language.

The following lists the six items that need to be observed when writing a C language program:
(1) As a convention, use lowercase letters to write a program.
(2) Separate executable statements in a program with a semicolon ";".
(3) Enclose execution units of functions or control statements with brackets "{" and "}"
(4) Functions and variables require type declaration.
(5) Reserved words cannot be used in identifiers (e.g., function names and variable names).
(6) Write comments between "/∗" and "∗/".

Programming Style

To improve program maintainability, programming conventions should be agreed upon by the programming team. Creating a template is a good way for the developers to establish a common programming style that will facilitate program development, debug and maintenance.
As shows in example of a programming style.

Note: Enclose a comment statement between "/*" and "*/".
Enclose a set of processing with brackets "{" and "}".

```
/*  Test Program  */
unsigned int ram1;
main()
{
    char a;                          Main Processing
    while(1) {
        if(a==ram1){
            break;}                  While Processing
        else{
            a=ram1;}
    }
}
```

Data Types

Constants" in C Language

Four types of constants can be handled in the C language: "integer", "real", "single character", and "character string".
This section explains the method of description and the precautions to be noted when using each of these constants.

Integer Constants

Integer constants can be written using one of three methods of numeric representation: decimal, hexadecimal, and octal. Below table shows each method for writing integer constants. Constant data are not discriminated between uppercase and lowercase.

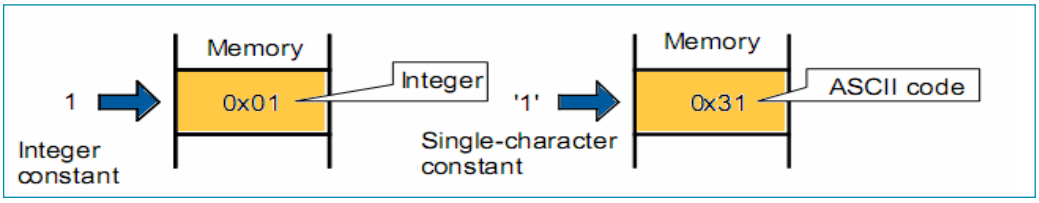
Method for Writing Integer Constants		
Numeration	**Method of Writing**	**Example**
Decimal	Normal mathematical notation (nothing added)	127, +127, -56
Hexadecimal	Numerals are preceded by 0x or 0X	0x3b, 0x3B
Octal	Numerals are preceded by 0 (Zero)	07, 041

Real Constants (Floating-Point Constants)

Floating-point constants refer to signed real numbers that are expressed in decimal. These numbers can be written by usual method of writing using the decimal point or by exponential notation using "e" or "E".
• Usual method of writing Example: 175.5, -0.007
• Exponential notation Example: 1.755e2, -7.0E-3

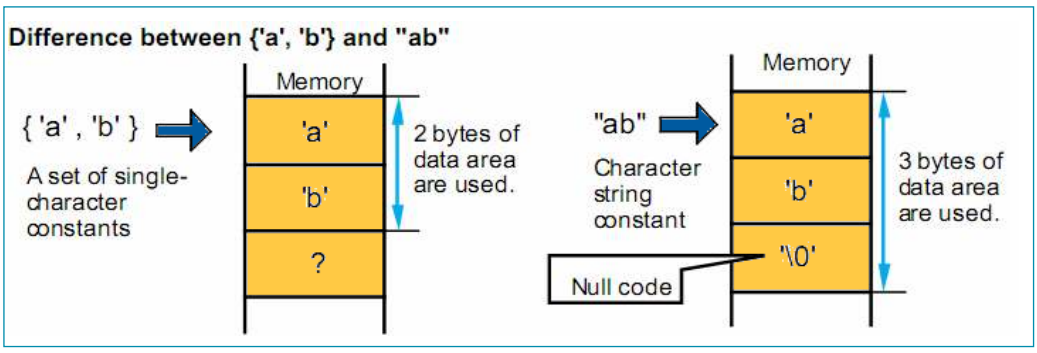
Single-Character Constants

Single-character constants must be enclosed with single quotations ('). In addition to alphanumeric characters, control codes can be handled as single-character constants. Inside the microcomputer, all of these constants are handled as ASCII code, as shown below diffrence between 1 and '1'.



Character String Constants

A row of alphanumeric characters or control codes enclosed with double quotations (") can be handled as a character string constant. Character string constants have the null character "\0" automatically added at the end of data to denote the end of the character string.

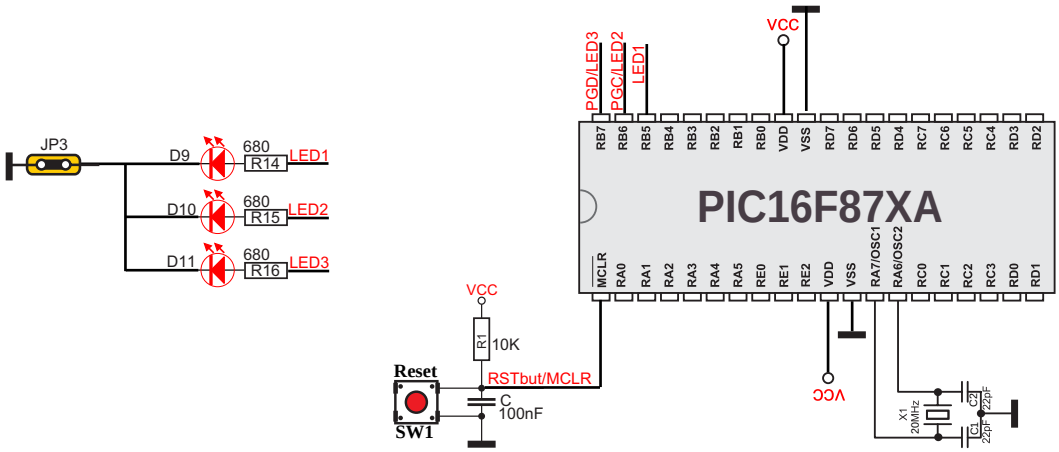


Lab:[1] Configuring I/O pins

The only purpose of this program is to turn on a few LED diodes on Port 'B' (RB5 pin, RB6 pin & RB7 pin). Figure below shows connection schematic.

In code all pin are set as digital, and set the direction output. In board there is external crystal 20Mhz.

As program execute Port 'B' direction is set as output & output as Low after that it enter in do while loop and it remain in that loop for infinity because while will remain always 1, after execution of program LED's ON after 1second than after 1 second become OFF and this process run for infinity.



output_x(value); // Outputs an entire byte to the port.
set_tris_x(value); // Sets the value of the I/O port direction register. A '1' is an input and '0' is for output.

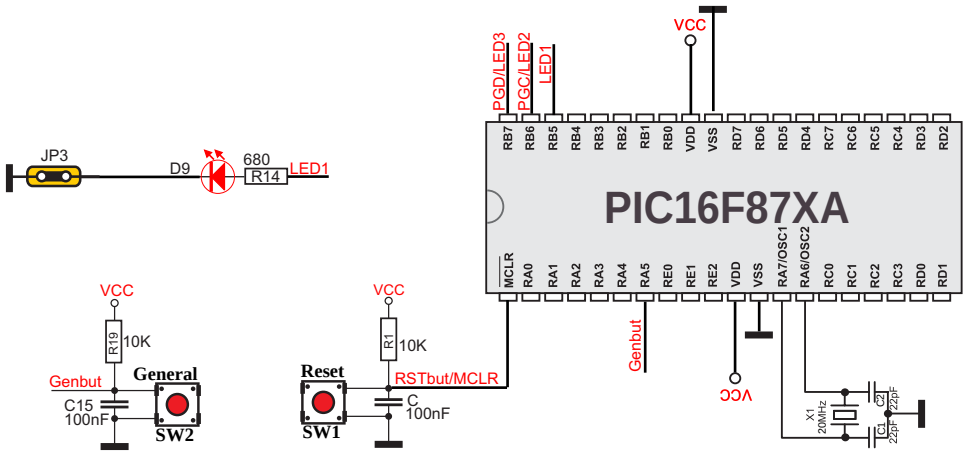
CODE:

```
/*
    • Project name:      LED Blinking
    • Revision History: 1.0
    • Description:
      This is a simple project. It turns on/off LED's connected to Port 'B' pins (RB5, RB6 & RB7)
    • Configuration:
      MCU:         PIC16F8XX
      Dev-Board: PICBox1
      Oscillator:  External 20 MHz
      Compiler:   CCS (Demo) for PIC
*/
```

```
#include <16F877A.h>                    // Define of micro which have used in circuit.
#use delay(oscillator=20000000)         // Compiler will set HS config bit (External 20MHz crystal used)
void main()
{
  set_tris_b(0x00);                     // Set Port 'B' direction to be output
  output_b(0x00);                       // Set Port 'B' as Low output
  while(1)                              // Endless loop
  {
    output_b(0x00);                     // Turn OFF LED's on Port 'B'
    delay_ms(1000);                     // 1 second delay
    output_b(0xFF);                     // Turn ON LED's on Port 'B'
    delay_ms(1000);                     // 1 second delay
  }
}
```

Lab:[2] Button Sensing

The only purpose of this program is to turn on/off LED, which are connected on Port 'B' pin RB5 by pressing push button which is connected with RA5 pin of Port 'A'. Figure below shows connection schematic.



In code all pin are set as digital, and set the direction output of Port 'B' & direction of Port 'A' as input. When switch press than pin goes to low stage (Zero) & **if** condition becomes true. Here in build function **input_x()** is used for sense the change stage of pins from which switch is connected button library used to sense the switch.

if (!input (PIN_A5)); // Condition become True when pin status change from High(1) to Low(0).
if (input (PIN_A5)); // Condition become True when pin status change from Low(0) to High(1).

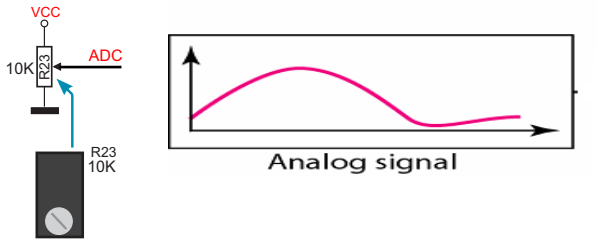
CODE:

```
/*
    • Project name:      Button Sensing
    • Revision History: 1.0
    • Description:
      This is a simple project. It turns on/off LED's connected to Port 'B' pin RB5 when switch is pressed
      which is connected with RB2 pin
    • Configuration:
      MCU:         PIC16F8XX
      Dev-Board: PICBox1
      Oscillator:  External 20 MHz
      Compiler:   CCS (Demo) for PIC
*/
```

```
#include <16F877A.h>                  // Define of micro which have used in circuit
#use delay(oscillator=20000000)       // User must manually set HS config bit (External 20MHz crystal used)
void main()
{
 set_tris_b(0x00);                    // Set Port 'B' direction to be output
 set_tris_a(0xFF);                    // Set Port 'A' direction to be input
 output_b(0xFF);                      // Set  Port 'B' as High output
 while(1)                             // Endless loop
 {
  If(!input(pin_A5))                  // Detect logical 0
  {
  while((!input(pin_A5)));            // For controlling de-bouncing
  output_toggle(pin_B5);              // Toggle state of pin
  }
 }
}
```

Lab:[3] Analog to Digital Conversion (ADC)

The PIC16F8XX A/D converter is used in this example. Is it necessary to mention that everything is rather simple?! A variable analog signal is applied to the AN0 pin, while the 10-bit result after conversion holds in variable **'data'** which is define as **float** in program.
In other words, the A/D converter always generates a 10-bit binary result, which means that it detects a total of 1024 voltage levels (2^{10} =1024).



The A/D result save in variable data you can use it for performing any function, or display the same over 16x2 LCD by adding LCD code and write that 'data' over LCD by using LCD functions etc. below for ADC conversion inbuilt function for ADC is used.

```
setup_adc(mode);          // Sets up the a/d mode like off, the ADC clock etc.
setup_adc_ports(value);   // Sets the available ADC pins to be analog or digital.
set_adc_channel(channel); // Specifies the channel to be use for the a/d call.
read_adc(mode);           // Starts the conversion and reads the value.
                          // The mode can also control the functionality.
```

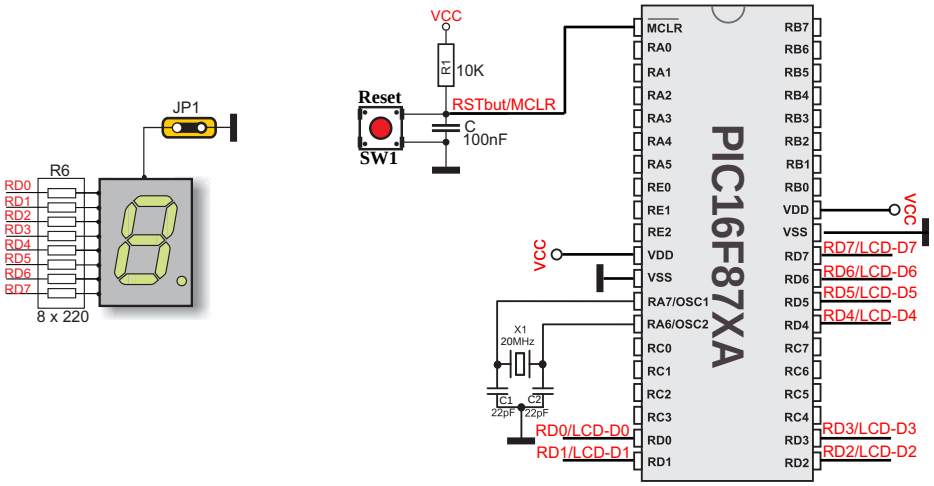
CODE:

```
/*
    • Project name:      Analog to Digital Conversion (ADC)
    • Revision History: 1.0
    • Description:
      This is a simple program to convert Analog signal to Digital and hold it in a variable for further use
    • Configuration:
      MCU:          PIC16F8XX
      Dev-Board:  PICBox1
      Oscillator:    External 20 MHz.
      Compiler:     CCS (Demo) for PIC.
*/
```

```
#include <16F877A.h >                  // Define of micro which have used in circuit.
#device ADC=10                         // 10-bit conversion
#fuses HS, NOWDT                       // Disable watchdog
#use delay(oscillator=20000000)        // Compiler will set HS config bit (External 20MHz crystal used)
void main()
{
  unsigned long  value=0;              // Variable define
  float data;                          // Variable define
  setup_adc_ports(ALL_ANALOG);         // Sets the available ADC pins to be analog or digital .
  setup_adc(ADC_CLOCK_INTERNAL);       // Sets up the a/d mode like off, the ADC clock etc.
  set_adc_channel(0);                  // Specifies the channel to be use for the a/d call
  While(1)                             // Endless loop
  {
    delay_ms(200);                     // 200 milli second delay
    value=read_adc();                  // Starts the conversion and reads the value
    data=value * (5.0/1023);           // Convert 10-bit  result
  }
}
```

Lab:[4] Seven Segment Based Timer

Here a single Common Cathode Seven Segment is parallel interface with PIC16F8XX through Port 'D'. Figure below shows connection schematic.
In code Port 'D' is set as digital output, and on the basis of Binary data over the Port 'D' output display over Seven Segment, & code is to display 0-9 digit over Seven Segment by the time difference of 1 second.



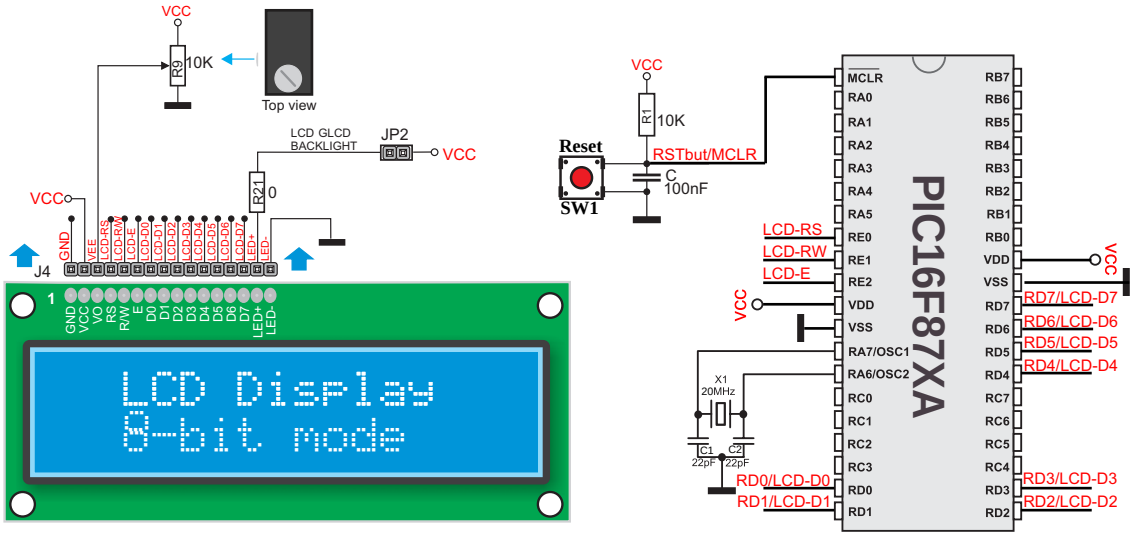
CODE:

```
/*
    • Project name:      Seven Segment Based Timer
    • Revision History: 1.0
    • Description:
      This is a simple program to use single Seven Segment as 1 second delay timer from 0-9
    • Configuration:
      MCU:         PIC16F8XX
      Dev-Board: PICBox1
      Oscillator:   External 20 MHz
      Compiler:    CCS (Demo) for PIC
*/
```

```
#include <16F877A.h>                    // Define of micro which have used in circuit
#use delay(oscillator=20000000)         // Compiler will set HS config bit (External 20MHz crystal used)
                                        // 0     1     2     3     4     5     6     7     8     9
unsigned int Seven_Segment[11]=         {0x7F, 0x41, 0xB3, 0xB6, 0xDC, 0xEE, 0xEF, 0x34, 0xFF, 0xFE};
int increment=0;                        // Variable define
void main()
{
  set_tris_d(0x00);                     // Set Port 'D' as output
  output_d(0x00);                       // Set  Port 'D' as High output
  delay_ms(5000);                       // Delay for 5 second
  while(1)                              // Endless loop
  {
    output_d(Seven_Segment[increment]); // Output of Port 'D' change with respect to increment value
    delay_ms(1000);                     // 1 second delay
    increment++;                        // Increment of increment variable by one after 1 second delay
    if(increment>=10)                   // Condition for checking increment value not equal or more than 10
    {
      increment=0;                      // Increment variable value set as 0
    }
  }
}
```

Lab:[5] Drive 16x2-LCD Display at 8-Bit mode

This example illustrates the use of an alphanumeric LCD display in 8- bit mode. 16x2 LCD data lines are interface with Port 'D' & command lines with Port 'E'. Figure below shows connection schematic.



This alphanumeric LCD can be driving in two modes 4-bit & 8-bit, here is an example code to display message over LCD at 8-bit mode.

CODE:

```
/*
    • Project name:      Drive 16x2- LCD Display at 8-Bit Mode
    • Revision History: 1.0
    • Description:
      This is a simple program to drive 16x2 LCD at 8-Bit Mode
    • Configuration:
      MCU:          PIC16F8XX
      Dev-Board:  PICBox1
      Oscillator:   External 20 MHz
      Compiler:     CCS (Demo) for PIC
*/
```

```
#include <16f877A.h>                    // Define of micro which have used in circuit
#use delay(oscillator=20000000)         // Compiler will set HS config bit (External 20MHz crystal used)
void command(unsigned char);            // LCD command function declare
void init_lcd();                        // LCD initialized function declare
void data(unsigned char);               // LCD data function declare

void main()
{
   set_tris_d(0x00);                    // Set  Port 'D' as output
   set_tris_e(0x00);                    // Set Port 'E' as output
   init_lcd();                          // LCD initialized function call
   command(0x84);                       // Command for selecting first row, fourth column
   data('E');data('l');data('e');data('S');data('o');data('f');      // Data to write over LCD
   command(0xC4);                                                   // Command for selecting second row, fourth column
   data('P');data('l');data('C');data('B');data('o');data('x');data('1');  // Data to write over LCD
   while(1);                                                                // Endless Loop
}
```

```
//Function for sending Command over LCD, in 8-Bit m
void command(unsigned char command_data)
{
   output_bit(PIN_E1,0);                  // R/W=0
   output_bit(PIN_E0,0);                  // RS=0
   output_D(command_data);                // Send command to LCD
   output_bit(PIN_E2,1);                  // E=1
   delay_ms(1);                           // 1 milli second delay
   output_bit(PIN_E2,0);                  // E=0
   delay_ms(10);                          // 10 milli second delay
}

//Function to initialized LCD, in 8-Bit mode
void init_lcd()
{
   command(0x38);                         // Function set 8-bit, 2-line, 5x7-dots.
   delay_ms(30);                          // 30 milli second delay
   command(0x0C);                         // Display on cursor off
   delay_ms(10);                          // 10 milli second delay
   command(0x06);                         // Entry mode
   delay_ms(10);                          // 10 milli second delay
   command(0x01);                         // Clear LCD
   delay_ms(5);                           // 5 milli second delay
}

//Function for sending Data over LCD, in 8-Bit mode
void data(unsigned char display_data)
{
   output_bit(PIN_E1,0);                  // R/W=0
   output_bit(PIN_E0,1);                  // RS=0
   output_D(display_data);                // Send data to LCD
   output_bit(PIN_E2,1);                  // E=1
   delay_ms(1);                           // 1 milli second delay
   output_bit(PIN_E2,0);                  // E=0
   delay_ms(1);                           // 1 milli second delay
}
```

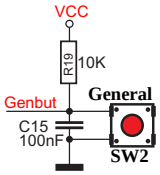
Note: For more details over PIC & CCS refer to user manual of PIC & CCS given with PICBox1.

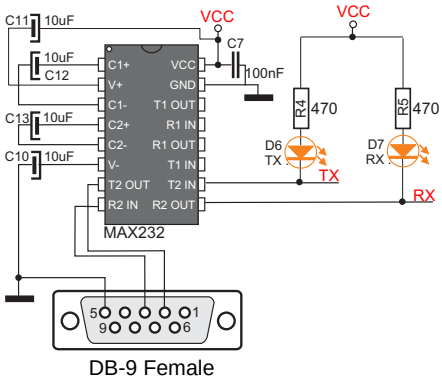
Lab:[6] Rs232 Communication

The example demonstrates a simple function of Rs232 communication of PICBox1 with PC, using in build function of CCS for Rs232 communication. When Switch which is connected with Port 'A' pin RA5, then a fix message "EleSof~PICBox1, Ver0.1" is sent over PC serial port and display over hyper-terminal. The controller pin RC6 is for transmit the data & RC7 for receive the data from interface device (PC, Microcontroller or Device which can communicate by using Rs232 protocol). Figure below shows connection schematic.

HyperTerminal setting for interface PICBox1 with PC.

- **Baud rate : 9600 (Change with respect to code)**
- **Data Bits : 8**
- **Parity : None**
- **Stop Bits : 1**
- **Flow Control : None**





Inbulid function used in code are:

```
#use rs232 (baud = --, xmit=PIN_--, rcv=PIN_--)   // Setting for Rs232 communication
printf("----");                                   // Send data through transmit pin
```

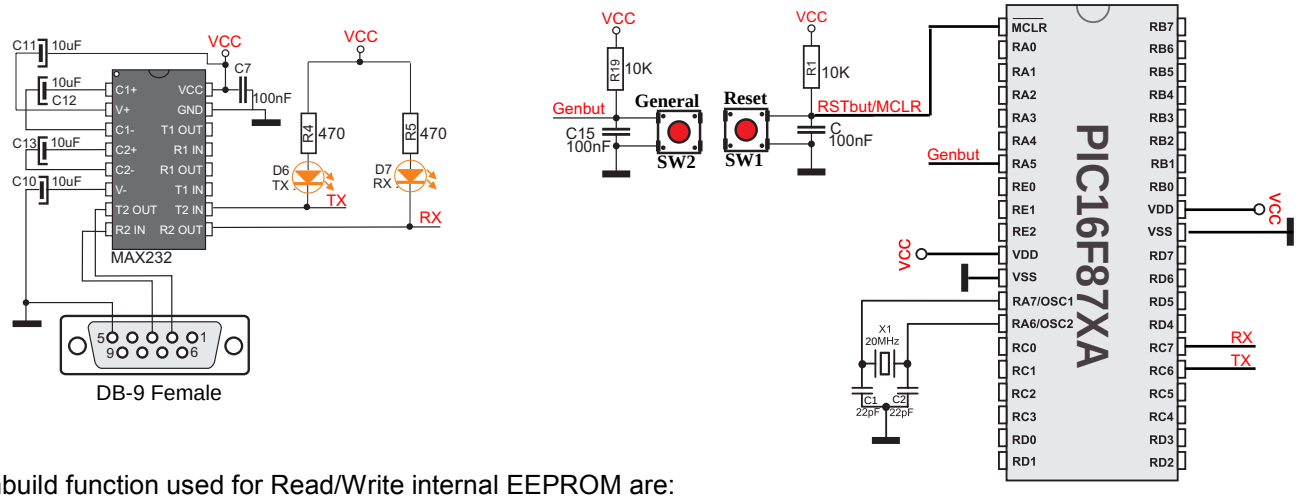
CODE:

```
/*
   • Project name:      Rs232 Communication
   • Revision History: 1.0
   • Description:
     This code demonstrates communication of PICBox1with PC using Rs232 protocol
   • Configuration:
     MCU:         PIC16F8XX
     Dev-Board: PICBox1
     Oscillator:   External 20 MHz
     Compiler:    CCS (Demo) for PIC
*/
```

```
#include <16F877A.h >                    // Define of micro which have used in circuit
#include <Stdio.h>                       // Standard Input/Output library define
#use delay(oscillator=20000000)          // Compiler will set HS config bit (External 20MHz crystal used)
#fuses HS, NOWDT                                  // Disable watchdog
#use rs232(baud = 9600, xmit =PIN_C6, rcv = PIN_C7)  // Setting for Rs232 communication
void main()
{
 set_tris_a(0xFF);                                // Set Port 'A' direction to be input
 while(1)                                         // Endless loop
 {
   If (!input(pin_A5))                            // Detect logical 0
  {
   while((!input(pin_A5)));                       // For controlling de-bouncing
   Printf("EleSof~PICBox1, Ver0.1");              // Send Data over transmit RC6
  }
 }
}
```

Lab:[7] Read/Write Internal EEPROM

This example illustrates write to and read from built-in EEPROM memory. The program works as follows. A data write to internal EEPROM memory at particular location address and again read by using EEPROM read function on pressing switch connected with RA5 pin of Port 'A', & that read data display over UART. Figure below shows connection schematic.



Inbuild function used for Read/Write internal EEPROM are:

write_eeprom(Address, Data); // Write data over define address.
read_eeprom(Address); // Read data from define address

Code:

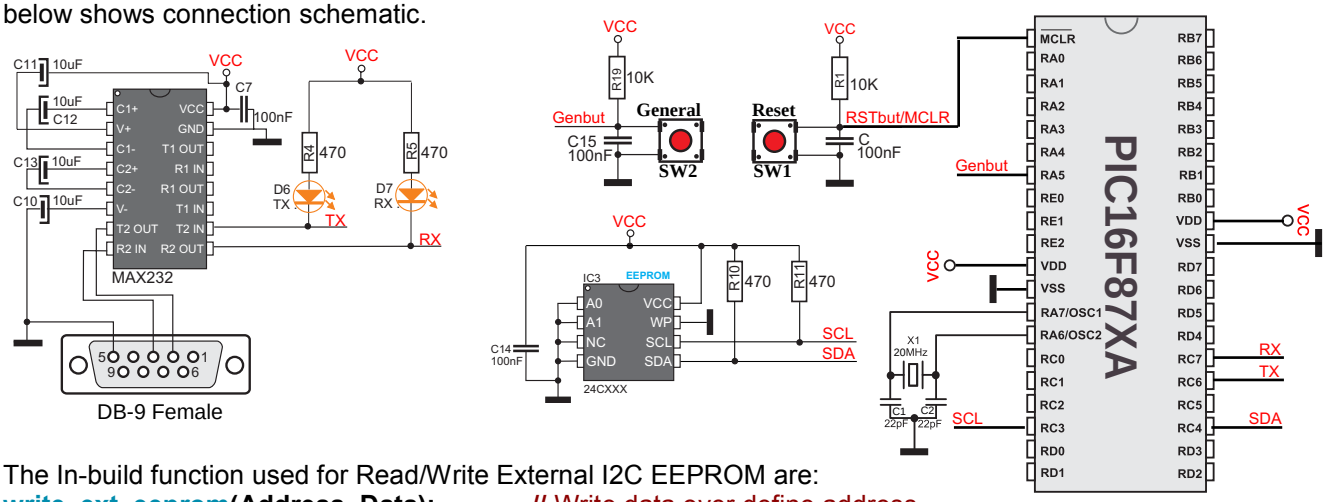
```
/*
    • Project name:      Read/Write Internal EEPROM
    • Revision History: 1.0
    • Description:
      This code demonstrates how to write data in internal EEPROM & read the data when switch pressed
    • Configuration:
      MCU:         PIC16F8XX
      Dev-Board: PICBox1
      Oscillator:  External 20 MHz
      Compiler:    CCS (Demo) for PIC
*/
```

```
#include <16F877A.h >                    // Define of micro which have used in circuit
#use delay(oscillator=20000000)          // Compiler will set HS config bit (External 20MHz crystal used)
#include <stdio.h>                                 // Standard Input/Output library define
#use rs232(baud = 9600, xmit =PIN_C6, rcv = PIN_C7) // Setting for Rs232 communication
void main()
{
 set_tris_a(0xFF);                                 // Set Port 'A' direction to be input
 while (1)                                         // Endless loop
 {
  If (!input(pin_A5))                              // Detect logical 0
  {
   while((!input(pin_A5)));                        // For controlling de-bouncing
   write_eeprom(0x0,123);                          // Write data in internal  EEPROM
   printf("%d", read_eeprom(0x00));                // Read data from internal EEPROM & send over UART
  }
 }
}
```

Note: Data should not be greater than Decimal (255) or Binary (8-bit)

Lab:[8] Read/Write External I2C EEPROM

This example illustrates write to and read from external I2C EEPROM memory. The program works as follows. A data write to external EEPROM memory at particular location address and again read by using EEPROM read function on pressing switch connected with RA5 pin of Port 'A', & that read data display over UART. Figure below shows connection schematic.



The In-build function used for Read/Write External I2C EEPROM are:

```
write_ext_eeprom(Address, Data);        // Write data over define address
read_ext_eeprom(Address);               // Read data from define address
```

Code:

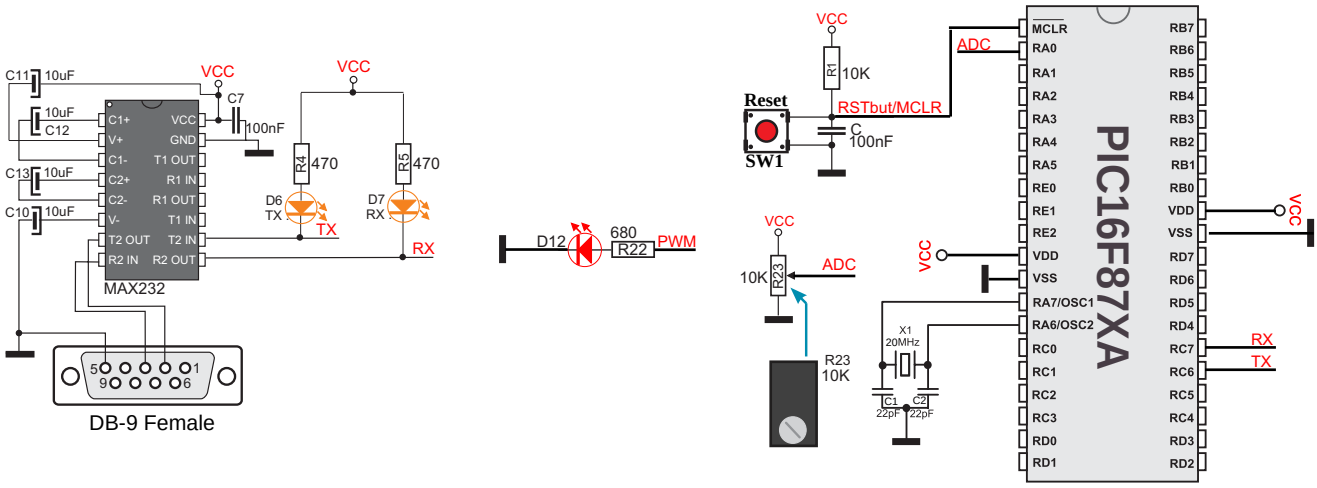
```
/*
    • Project name:      Read/Write External I2C EEPROM
    • Revision History: 1.0
    • Description:
      This code demonstrates how to read or write data over & from external I2C EEPROM
    • Configuration:
      MCU:        PIC16F8XX
      Dev-Board: PICBox1
      Oscillator:  External 20 MHz
      Compiler:   CCS (Demo) for PIC
*/
```

```
#include <16F877A.h >                          // Define of micro which have used in circuit
#use delay(oscillator=20000000)                // Compiler will set HS config bit (External 20MHz crystal used)
#define EEPROM_SDA PIN_C4                      // SDA pin define
#define EEPROM_SCL PIN_c3                      // SCL pin define
#include <2416.c>                              // Library define for 16K I2C EEPROM (24Cxx)
#include <stdio.h>                                     // Standard Input/Output library define
#use rs232(baud = 9600, xmit =PIN_C6, rcv = PIN_C7)  // Setting for Rs232 communication
void main()
{
 set_tris_a(0xFF);                                     // Set Port 'A' direction to be input
 while (1)                                             // Endless loop
 {
  If (!input(pin_A5))                                  // Detect logical 0
  {
   while((!input(pin_A5)));                            // For controlling de-bouncing
   write_ext_eeprom(0x0301,0x05);                      // Write data in internal  EEPROM
   printf("%d", read_ext_eeprom(0x0301));              // Read data from internal EEPROM & send over UART
  }
 }
}
```

Note: Data should not be greater than Decimal (255) or Binary (8-bit)

Lab:[9] PWM (Pulse Width Modulation)

This example illustrates the use of CCP1 module in PWM mode to control blinking rate of LED connected with RC2 pin of Port 'C'. To make things more interesting, the duty cycle of pulse is change by using analog signal as digital value after Analog to Digital Conversion & frequency of pulse is change by changing the value of Timer2, for analog signal a variable resistor which is connected with RA0 pin is used. Figure below shows connection schematic.



The In-build function used for setup PWM, Duty Cycle & Timer for Frequency of output pulse are:

```
setup_ccp1(mode);          //  Sets the mode to capture, compare or PWM, for PWM mode=(CCP_PWM)
set_pwm1_duty(value);      //  The value is written to the pwm1 to set the duty cycle.
setup_timer_2 (mode, period, postscale);  // Setup timer 2
```

Note:

- **mode** may be one of: T2_DISABLED, T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16
- **period** is a int 0-255 that determines when the clock value is reset
- **postscale** is a number 1-16 that determines how many timer overflows before an interrupt: (1 means once, 2 means twice)

Calcualtion:

- If **value** is **INT** then calculation **value** for **PWM Duty Cycle:** **(value*(1/clock)*t2div)**
- If **value** is **LONG INT** then calculation **value** for **PWM Duty Cycle:** **(value*4*(1/clock)*t2div)**
- The **Timer_2 Cycle Time calculation :** **(1/clock)*4*t2div*(period+1)**

In this program **clock=20000000** and **period=127** (below) **& postsacle=1** then result of timer cycle is:
(1/20000000)*4*16*128= 409.6 us or 2.4 khz

Code:

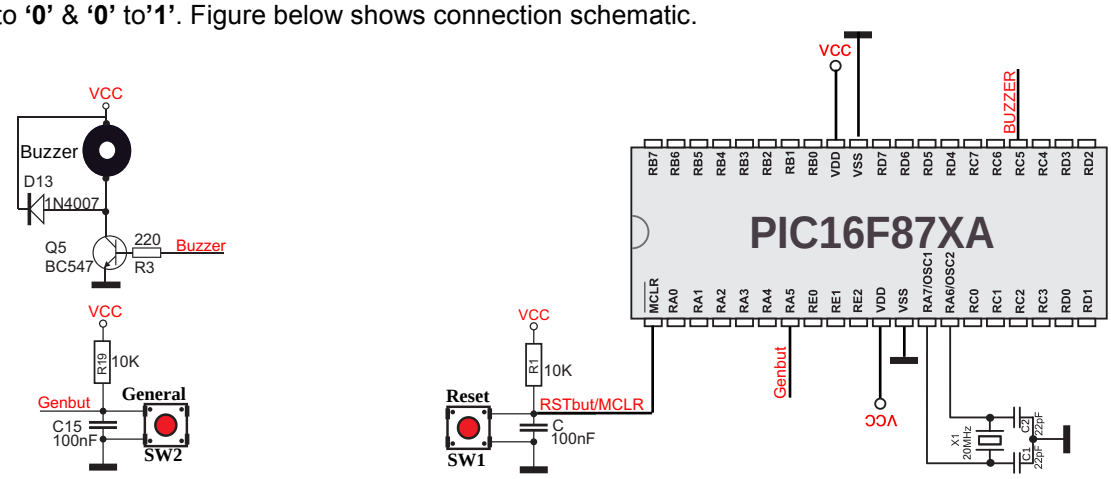
```
/*
    • Project name:      PWM (Pulse Width Modulation)
    • Revision History: 1.0
    • Description:
      This code demonstrates to control the blinking rate of LED using PWM
    • Configuration:
      MCU:          PIC16F8XX
      Dev-Board: PICBox1
      Oscillator:   External 20 MHz
      Compiler:     CCS (Demo) for PIC
*/
```

```
#include <16F877A.h>                          // Define of micro which have used in circuit
#device ADC=10                                // 10-bit conversion
#fuses HS,NOWDT,NOPROTECT,NOLVP               // Disable watchdog
#use delay(oscillator=20000000)               // Compiler will set HS config bit (External 20MHz crystal used)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)  // Setting for Rs232 communication
void main()
{
  byte value;                                 // Variable define
  setup_ccp1(CCP_PWM);                        // Configure CCP1 as a PW
  setup_timer_2(T2_DIV_BY_16, 127, 1);        // In this program clock=20000000, period=127 & postscale=1
                                              // The cycle time will be (1/clock)*4*t2div*(period+1)
                                              // (1/20000000)*4*16*128= 409.6 us or 2.4 khz
  setup_adc_ports(ALL_ANALOG);                // Sets the available ADC pins to be analog or digital
  setup_adc(ADC_CLOCK_INTERNAL);              // Sets up the a/d mode like off, the ADC clock etc.
  set_adc_channel(0);                         // Specifies the channel to be use for the a/d call
  while( TRUE )                               // Endless loop
  {
    value=read_adc();                         // Value read from ADC conversion
    printf("%2X\r",value);                    // Send ADC value over UART
    set_pwm1_duty(value);                     // This sets the time the pulse is
                                              // If value is LONG INT: value*(1/clock)*t2div
                                              // If value is INT: value*4*(1/clock)*t2div

  }
}
```

Lab:[10] Drive Buzzer

The only purpose of this program is to turn on/off Buzzer, which are connected on Port 'C' pin RC5 by using transistor as switch. On pressing push button which is connected with RA5 pin of Port 'A' the status of pin toggle from **'1'** to **'0'** & **'0'** to**'1'**. Figure below shows connection schematic.



In code all pin are set as digital, and set the direction output of Port 'C' & direction of Port 'A' as input. When switch press than pin goes to low stage (Zero) & **if** condition becomes true. Here in build function **input_x()** is used for sense the change stage of pins from which switch is connected button library used to sense the switch.

```
if ( !input (PIN_A5) );      // Condition become True when pin status change from High(1) to Low(0).
if ( input (PIN_A5) );       // Condition become True when pin status change from Low(0) to High(1).
```

CODE:

```
/*
    • Project name:      Drive Buzzer
    • Revision History: 1.0
    • Description:
      This is a simple project. To drive Buzzer on pressing switch connected with controller
      Configuration:
      MCU:         PIC16F8XX
      Dev-Board: PICBox1
      Oscillator:  External 20 MHz
      Compiler:    CCS (Demo) for PIC
*/
```

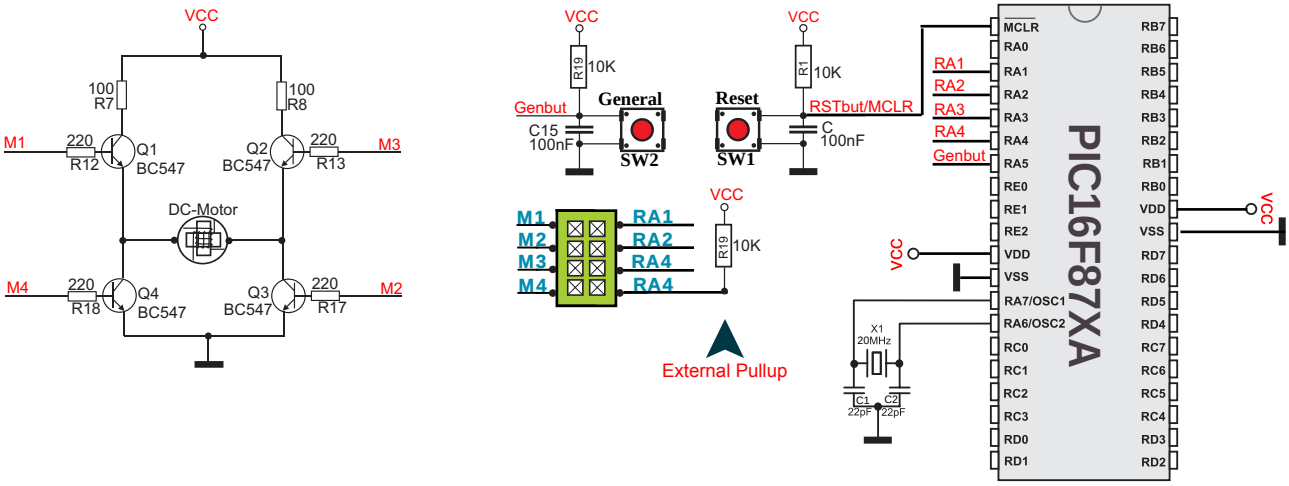
```
#include <16F877A.h>                    // Define of micro which have used in circuit
#use delay(clock=20000000)              // User must manually set HS config bit (External 20MHz crystal used)
void main()
{
 set_tris_c(0x00);                      // Set Port 'C' direction to be output
 set_tris_a(0xFF);                      // Set Port 'A' direction to be input
 output_c(0x00);                        // Set  Port 'C' as High output
 while (1)                              // Endless loop
 {
  If (!input (pin_A5))                  // Detect logical 0
  {
   while ((!input (pin_A5)));           // For controlling de-bouncing
   output_toggle(pin_C5);               // Toggle state of pin
  }
 }
}
```

Lab:[11] Drive Dc Motor

In this lab we will see that how to drive & control Dc Motor, interface with controller. The purpose of program to make Dc Motor **ON/OFF** & control its rotation **Clockwise/Anticlockwise**. On pressing first time push button which is connected with RA5 pin of Port 'A' the Dc Motor become **ON/Clockwise,** when second time push button pressed Dc Motor start rotating in **Anticlockwise** direction & on third time press of push button Dc motor become **OFF**. The Dc Motor interface with Controller using Transistor H-Bridge arrangement.

Dc Motor Status on pressing Push Button

Push Button Pressed	Pin-RA1	Pin-RA2	Pin-RA3	Pin-RA4	Dc Motor Status
Initial Stage	0	0	0	0	OFF
First Time Pressed	1	1	0	0	ON/Clockwise
Second Time Pressed	0	0	1	1	Anticlockwise
Third Time Pressed	Mov=0 (Initial Stage)				OFF



Note: RA4 is a Open Drain when configure as Output, so when configure as output pullup this with 10K resistor .

Here in build function **output_bit(pin, value)** is used for changing the status of particular pin of any port without effecting other pins of that port. In function **value** can be set as **0** or **1** & in function **pin** is whose status have to change.

output_bit(pin, value) // For changing particular pin status by setting value as 0 or 1

CODE:

```
/*
    • Project name:      Drive Dc Motor
    • Revision History: 1.0
    • Description:
      This is a simple project, to drive & control Dc Motor ON/OFF & Clockwise/Anticlockwise direction
    • Configuration:
      MCU:        PIC16F8XX.
      Dev-Board: PICBox1.
      Oscillator:  External 20 MHz
      Compiler:   CCS (Demo) for PIC
*/
```

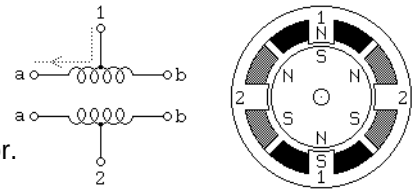
```
#include <16F877A.h>                    //Define of micro which have used in circuit
#use delay(oscillator=20000000)         // Compiler will set HS config bit (External 20MHz crystal used)
void main()
{
  int mov=0;                            // Define variable
  setup_adc_ports(NO_ANALOGS);          // All pins digital
  set_tris_a(0x20);                     // Set Port 'A' pins 0-4 as output & pin 5 as input
  output_a(0x00);                       // Set Port 'A' as Low output
  while (1)                             // Endless loop
  {
    If (!input(pin_A5))                 // Detect logical 0
    {
      while((!input(pin_A5)));          // For controlling de-bouncing
      mov++;
    }
    if(mov==0)                          // Condition for OFF
    {
      output_bit( PIN_A1, 0);           // Pin A1 set low
      output_bit( PIN_A2, 0);           // Pin A2 set low
      output_bit( PIN_A3, 0);           // Pin A3 set low
      output_bit( PIN_A4, 0);           // Pin A4 set low
    }
    if(mov==1)                          // Condition for ON/Clockwise direction
    {
      output_bit( PIN_A1, 1);           // Pin A1 set high
      output_bit( PIN_A2, 1);           // Pin A2 set high
      output_bit( PIN_A3, 0);           // Pin A3 set low
      output_bit( PIN_A4, 0);           // Pin A4 set low
    }
    if(mov==2)                          // Condition for Anticlockwise direction
    {
      output_bit( PIN_A1, 0);           // Pin A1 set low
      output_bit( PIN_A2, 0);           // Pin A2 set low
      output_bit( PIN_A3, 1);           // Pin A3 set high
      output_bit( PIN_A4, 1);           // Pin A4 set high
    }
    if(mov==3)                          // Condition for OFF
    {
      mov=0;                           // Variable value set zero
    }
  }
}
```

Lab:[12] Drive Stepper Motor

In this lab we will see that how to drive & control Stepper Motor, interface with controller. The purpose of program to make Stepper Motor **ON/OFF** & control its rotation **Clockwise/Anticlockwise**. On pressing first time push button which is connected with RA5 pin of Port 'A' the Stepper Motor become **ON/Clockwise,** when second time push button pressed Stepper Motor start rotating in **Anticlockwise** direction & on third time press of push button Stepper motor become **OFF**. The Stepper Motor interface with Controller using ULN2003.

Stepper Motor Status on pressing Push Button

Push Button Pressed	Stepping Mode	Stepper Motor Status
Initial Stage	High Torque Stepping	OFF
First Time Pressed		ON/Clockwise
Second Time Pressed		Anticlockwise
Third Time Pressed		OFF



Stepping-Modes

There are several stepping modes that you can use to drive the stepper motor.

Pulse	Coil a1	Coil b1	Coil a2	Coil b2
1	ON			
2		ON		
3			ON	
4				ON

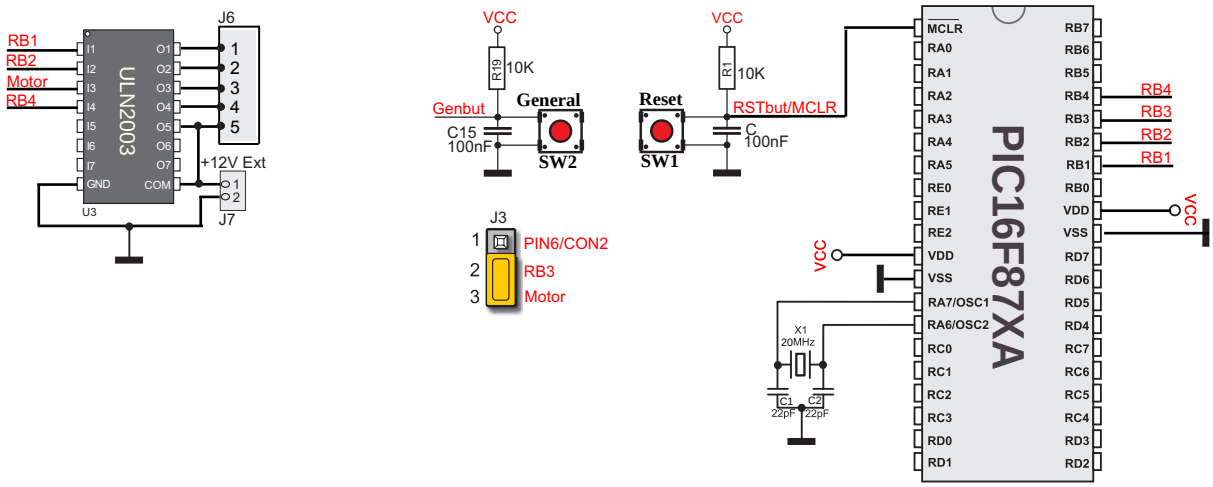
Single Stepping - the simplest mode turns one coil ON at a time. 48 pulses are needed to complete one revolution. Each pulse moves rotor by 7.5 degrees. The following sequence has to be repeated 12 times for motor to complete one revolution.

Pulse	Coil a1	Coil b1	Coil a2	Coil b2
1	ON	ON		
2		ON	ON	
3			ON	ON
4	ON			ON

High Torque Stepping - high power / precision mode turns ON two coils on at a time. 48 pulses are needed to complete one revolution. Each pulse moves rotor by 7.5 degrees. The following sequence has to be repeated 12 times for motor to complete one revolution.

Pulse	Coil a1	Coil b1	Coil a2	Coil b2
1	ON			
2	ON	ON		
3		ON		
4		ON	ON	
5			ON	
6			ON	ON
7				ON
8	ON			ON

Half Stepping - stepping is doubled and motor needs 96 pulses to complete one revolution. Each pulse moves rotor by approximately 3.75 degrees. Notice the mix of single stepping mode (lighter green) and high torque mode (darker green).



CODE:

```
/*
    • Project name:      Drive Stepper Motor
    • Revision History: 1.0
    • Description:
      This is a simple project, to drive & control Stepper Motor ON/OFF & Clockwise/Anticlockwise direction
    • Configuration:
      MCU:        PIC16F8XX.
      Dev-Board: PICBox1.
      Oscillator:  External 20 MHz
      Compiler:   CCS (Demo) for PIC
*/
```

```
#include <16F877A.h>                    // Define of micro which have used in circuit
#use delay(oscillator=20000000)         // Compiler will set HS config bit (External 20MHz crystal used)
void main()
{
  int mov=0;                            // Define variable
  set_tris_a(0xFF);                     // Set Port 'A' pins as output
  set_tris_b(0x00);                     // Set Port 'B' pins  as output
  output_b(0x00);                       // Set Port 'B' as Low output
  while (1)                             // Endless loop
  {
    If(!input (pin_A5))                 // Detect logical 0
    {
      while((!input(pin_A5)));          // For controlling de-bouncing
      mov++;                            // Increment of integer
    }
    if(mov==0)                          // Condition for OFF
    {
      output_b(0x00);                   // Set Port 'B' as Low output
    }
    if(mov==1)                          // Condition for ON/Clockwise direction
    {
      output_b(0x02);  delay_ms(50);    // Pin B1 set high & Delay of 50ms
      output_b(0x04);  delay_ms(50);    // Pin B2 set high & Delay of 50ms
      output_b(0x08);  delay_ms(50);    // Pin B3 set high & Delay of 50ms
      output_b(0x10);  delay_ms(50);    // Pin B4 set high & Delay of 50ms
    }
    if(mov==2)                          // Condition for Anticlockwise direction
    {
      output_b(0x10);  delay_ms(50);    // Pin B4 set high & Delay of 50ms
      output_b(0x08);  delay_ms(50);    // Pin B3 set high & Delay of 50ms
      output_b(0x04);  delay_ms(50);    // Pin B2 set high & Delay of 50ms
      output_b(0x02);  delay_ms(50);    // Pin B1 set high & Delay of 50ms
    }
    if(mov==3)                          // Condition for OFF
    {
      mov=0;                            // Variable value set zero
    }
  }
}
```

EleSof®

Development System

DISCLAIMER

This product is owned by the EleSof are protected by copyright law. Therefore, this manual is to be treated as any other copyright material. No part of this manual, including product & software described herein, may be reproduces, store in a retrieval system, translated or transmitted in any form or by any means, without the prior written permission of EleSof. The manual PDF edition can be printed for private or local use, but not for distribution. Any modification of this manual is prohibited.

EleSof provides this manual 'as is' without warranty of any kind, either expressed or implied, including , but not limited to, the implied warranties or condition of merchantability or fitness for a particular purpose.

Elesof shall assume no responsibility or liability for any errors, omissions & inaccuracies that may appear in this manual. In no event shall EleSof, its directors, officers, employees or distributors be liable for any indirect, specific, incidental or consequential damages (including damages for loss of business profits & business information, business interruption or any other pecuniary loss) arising out of the use of this manual or product, even if EleSof has been advised of the possibility of such damages. EleSof reserves the right to change information contained in this manual at any time without prior notice, if necessary.

All the product, tools & corporate names appearing in this kit & manual may or may not registered trademarks or copyright of their respective company, & are only used to identification or explanation & to the owners' benefit, with no intent to infringe.

EleSof®

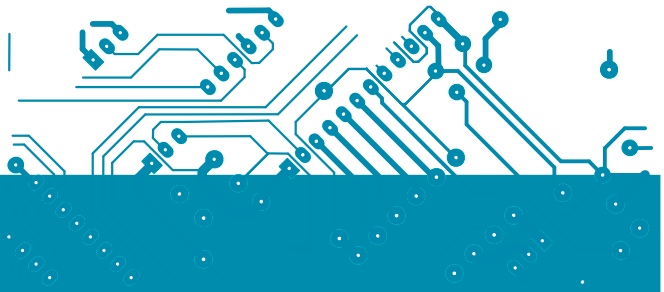
Development System

HIGH RISK ACTIVITIES

The products of EleSof are not fault - tolerant nor designed, manufactured or intended for use or resale as on – line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication system, air traffic control, direct life support machines or weapons system in which the failure of software could lead directly to death, personal injury or severe physical or environment damage ('High Risk Activities). EleSof & its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Copyright 2010-2012 by EleSof Technologies Pvt. Ltd. All right reserved.

EleSof®
Development System



If you want to learn more about our products, please visit our website at
www.elesof.com & www.projectsmaker.com

If you have any questions, comments or business proposals, do not hesitate to contact us.

If you are experiencing some problems with any of our products or just need additional information, please place write mail or call to us at
contact@elesof.com & info@elesof.com
+91-9911-7095-92 & +91-9044-1359-01